The Evolutionary Design Model (EDM)
For the Design of Complex Engineered Systems
MASDAR City as a Case Study

By
Anas Alfaris

Ph.D. Design and Computation
Massachusetts Institute of Technology, 2009

Master of Science in Architecture Studies
University of Pennsylvania, 2002

Master of Architecture and Building Technology
University of Pennsylvania, 2000

Bachelor of Science in Architecture and Building Engineering
King Saud University, 1998

Submitted to the School of Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computation for Design and Optimization

At the

Massachusetts Institute of Technology
September 2009

Signature of author .....

School of Engineering, MIT
August 7th, 2009

Certified by: .....

Oliver de Weck
Associate Professor of Aeronautics and Astronautics
and Engineering Systems
Department of Aeronautics and Astronautics
and Engineering Systems Division, MIT

Accepted by: .....

Jaime Peraire
Professor of Aeronautics and Astronautics
Director, Computation for Design and Optimization Program

The Evolutionary Design Model (EDM)
For the Design of Complex Engineered Systems
MASDAR City as a Case Study

By Anas Alfaris

Submitted to the School of Engineering on August 7th, 2009
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computation for Design and Optimization

Abstract

This thesis develops a framework for constructing an Evolutionary Design Model (EDM) that would enhance the design of complex systems through an efficient process. The framework proposed is generic and suggests a group of systematic methodologies that eventually lead to a fully realized and integrated design model. Within this model, complexities of the design are handled and the uncertainties of the design evolution are managed. Using the framework, vast design spaces can be searched while solutions are intelligently modified, their performance evaluated, and their results aggregated into a compatible set for design decisions.

The EDM is composed of several design states as well as design evolving processes. A design state describes a design at a particular point in time and maps the system's object to the system's requirements and identifies its relation to the context in which the system will operate. A design evolving process involves many sub-processes which include formulation, decomposition, modeling, and integration. These sub-processes are not always carried out in a sequential manner, but rather a continuous move back and forth to previous and subsequent stages is expected. The resulting design model is described as an evolutionary model that moves a system's design from simple abstract states to more complex and detailed states throughout its evolution.

The framework utilizes system modeling methodologies that include both logical and mathematical modeling methods. The type of model used within the EDM's evolving processes is highly dependent on and driven by design needs of each process. As the design progresses a shift from logical models to mathematical models occurs within the EDM.

Finally, a partial EDM is implemented within the context of a computational design system for Masdar city to demonstrate the application of the proposed framework.

Thesis Advisor: Oliver de Weck
Title: Associate Professor of Aeronautics and Astronautics and Engineering Systems

Acknowledgment

This thesis was a shared experience in which several professors, collaborators, friends and family members have had a part in. I would first like to thank my advisor, Professor Oliver de Weck, whose exceptional experience with multi-disciplinary optimization and the design of complex engineered systems had shaped my work and research. Oli has constantly provided me with valuable advice and always believed in my ideas and work. Furthermore, I cannot thank him enough for opening wide domains of research for me, whether at the System Engineering Division or the Strategic Engineering Group.

The work presented in this thesis on Masdar city was a joint effort with several other Collaborators. I would like to thank Professor Davor Svetinovic for his continuous support and encouragement throughout this research. I would also like to thank Professor Dov Dori for introducing me to the fascinating world of OPM. I would also like to thank Ariane Chepko who helped develop several of the synthesis and analysis modules. In addition, I would like to acknowledge Athar Sayed who helped in the decomposition of several of the city's systems. Furthermore, I would like to mention Charbel Risk for our several stimulating discussions.

I would especially like to thank Laura Koller for always keeping me on schedule and for her amazing support and patience with my endless delays in submitting this thesis. I would also like to thank the Masdar Institute of Science and Technology (MIST) for their support and funding of this research.

I have been blessed with great friends throughout my life, some of whom contributed a great deal to the work presented in this thesis. From those I would like to thank Saud Sharaf for being a great friend at MIT and beyond and for always being there when I needed him, especially when he offered to edit parts of this thesis at a very critical time. I would like to acknowledge Maher Elkhaldi who also was kind enough to help me edit a few chapters of this thesis on very short notice. I should also thank my dear friend Ahmed Rashad without whose help in formatting this thesis, I could not have finished on time.

I have also received enormous support from my family. I would like to first thank my siblings Nasreen, Eyas and especially my youngest sister Sawsan for their endless support and encouragement throughout this endeavor. I would also like to thank my dear wife Dr. Manal Alaamery who has always been my strongest supporter and my closest friend and companion. I would also like to thank my dear son Faris and daughter Aseel for their exceptional patient throughout my academic adventures. Finally I would like to thank my parents Architect Faris Alfaris and Professor Haya Alrawaf for their infinite prayers and guidance.

Anas Alfaris

Table of Contents

The Evolutionary Design Model (EDM) for the Design of Complex Engineered Systems
MASDAR City as a Case Study

1. Introduction

Recent science and technology advances have resulted in an increase in the complexity of engineered systems. A complex system is defined by Crawley et al. (2004) as a system that comprises components and interconnections, interactions, or interdependencies, all of which are hard to describe, understand, predict, manage, design, or change.

These engineered systems are composed of many subsystems that exhibit complex behaviors. Many of these behaviors are emergent as a result of nonlinear spatio-temporal interrelations between the subsystems at different levels of abstraction and organization. Complexity within engineered systems also implies that different subsystems are coupled so that changes in a certain subsystems may affect other subsystems.

The design of complex systems is not a well defined field. In the following sections I will discuss topics that relate to the design of complex systems to help develop a basic foundation and theoretical discourse. I will investigate topics such as Design Engineering, Systems Theory, and Computational modeling. Along this thesis, knowledge from different domains will be sought to support the presented approach.

1.1 Design Engineering

Design involves solving what Herbert Simon terms an ill-structured problem (Simon, 1973). An ill-structured problem is

one that cannot be solved by a linear chain of reasoning derived from the problem statement. Furthermore, it might not have a unique solution but a multiplicity of solutions. These design problem characteristics imply the need for many assumptions within the design process that can only be verified after a solution is reached. Given the numerous inputs that feed into design, it is not surprising that design presents a technical challenge even for relatively well understood products (Eppinger and Gebala, 1991).

Design is a complex activity requiring knowledge spanning many different domains. Even the most rudimentary design activities demand scientific knowledge, engineering skills and artistic creativity. Many real-world design problems cannot be modeled by one single model. Systems like aircrafts, automobiles or skyscrapers consist of continually and mutually interacting subsystems. The behavior of such complex systems is controlled by a variety of physical phenomena that are analyzed by different disciplines and that interact at the same time. Therefore, they require using groups of complementary tools and models that integrated together can describe the whole process (Yilmaz and Oren, 2004; Zeigler et al., 2000).

Designing is an activity that occurs with the prospect that the designed system will operate in both the natural and social worlds. Both these worlds introduce constraints on the variables and their associated values. As a result, design in this context can be viewed as a goal-oriented, constrained, decision making activity (Gero, 1990).

As stated earlier, design of complex systems depends on diverse design and engineering domains; these design problems usually comprise conflicting objectives. This constitutes another design challenge. To overcome this there is a need to support the rapid generation and evaluation of design alternatives to provide satisfactory design space search.

It was argued by Simon (1973) that a science of design could exist someday where design can be discussed in terms of well-established theories and practices. Simon claimed that design should move through multiple stages from its then current pre-science stage in order to become a mature science. He described that mature stage as acquiring a state of discipline where a consistent body of scientific research and practice exists and encompasses law, theory, application, and instrumentation (Kuhn, 1970).

Dixon (1987) argued that in engineering design particularly, both education and practice are led mostly by expert empiricism and intuition and specialized experience without sufficient scientific foundation. He stated that design in this case is very different from disciplines such as physics, chemistry, or biology where theories can be tested by controlled experiments. He argued that design was more complex than other fields because it involved not only people and organizations, but also the natural physical world and the in-progress design, which refers to the to-be-manufactured-sold-and-used system. This complexity also lies in design being a process, where processes are not the typical subjects of theoretical formulations.

Hongo (1985) defines a design science or a scientific study of design activities as a collection of logically connected knowledge such as design methodology and design technique. A detailed definition implies that design theory is a system of methodical rules that identify the procedures possibly expected to conduct a planned route towards achieving a desired goal. He classifies types of rules according to methods of thinking, such as intuitive or discursive, and according to goals and applications, such as methods for solution search, evaluation, and calculation.

Coyne et al. (1990) provide another definition of science and design. As opposed to science, which formulates knowledge

through deriving relationships between observed phenomena, design can be described as an action that starts with intentions and uses available knowledge to reach a specific entity whose properties should meet those original intentions. The role of design is defined as that which utilizes that knowledge to transform a formless description into a specific description of form. This description, known as the design solution, is generated pragmatically according to the capacity of knowledge available to the designer. It is a compromise, rather than an ideal or correct solution, that meets to some extent the original intentions.

Design therefore can be seen as an evolutionary process that evolves over time. Such a process can assist in handling design complexity by breaking the design into stages that move from the simple and abstract to the more complex and concrete.

1.2 Systems Theory

Recently systems theory also provided a significant view on the process of system design. It provides a framework for the description of several groups and objects that act in concert to produce some result. It investigates the principles common to all complex entities and the models which can be used to describe them.

Papalambros and Wilde (2000) define a *system* as a collection of entities that perform a specified set of tasks. For example, an automobile is a system that transports passengers. Schmidt and Taylor (1970) define a *system* as a collection of entities, such as people or machines, which act and interact together toward the accomplishment of some logical end.

Purposeful action is a key feature of any system. Sage and Armstrong (2000) define a system as a group of components that work together for a specified purpose. This implies that any system has to perform specific tasks that achieve its

purposes. A system can be seen as consisting of a group of entities that affect one another within an environment to build up a larger pattern that is different from any of those initial entities.

A system is shown to be very perspective-dependent, where different components of the system could be grouped according to different perspectives to build up different notions of systems (Sage and Armstrong, 2000). In the engineering of the system, it is thus important to carefully define the nature of the system, its exact scope of components as well as the interfaces to it.

Law and Kelton (1999) show that, in practice, the objectives of a particular study determine what is meant by a *system*. The collection of entities that constitute a system for a specific study may be only a subset of the overall system for another. Law and Kelton thus define what is called the *state* of the system, which is that group of variables that describes a system relative to the objectives of a study at a particular time.

A system usually operates under causality, where the system tasks are performed due to some kind of stimulus or input (Papalambros and Wilde, 2000). This implies that these inputs have a significant effect on the system behavior. What actually constitutes an input or output relies primarily on the viewpoint from which the system is examined. Each systems viewpoint in general is based on a specific level of knowledge of the components of the system and its internal structure, the complexity of the system performance in relation to the environment, in addition to other engineering and management issues. A system is analyzed at a specific level of complexity that corresponds to the interests of the individuals studying it.

1.3 Computational Simulation

Recent possibilities facilitated by advancements in computational power and new developments in computer-based modeling and numerical methods, such as finite element analysis (FEA), computational fluid dynamics (CFD), visualization, process simulation and others, have enabled the simulation of design performance in virtual environments. This provided designers and engineers with information that can assist in some decision making (Paydarfar, 2001). However, these design models are usually discipline specific and hence lack the ability to supply sufficient understanding of the possible tradeoffs between different disciplines. Therefore the design insight gained through these tools and technologies remains limited to a single domain and therefore, their potential to enhance and inform the design process of complex systems has not been fully realized.

1.4 Thesis Structure

As the title suggests, this thesis attempts to develop a framework for constructing an Evolutionary Design Model (EDM) that would enhance the design of complex systems through an efficient process.

The framework proposed is generic and suggests a group of systematic methodologies that eventually lead to a fully realized and integrated design model. Within this model, complexities of the design are handled and the uncertainty of the design evolution is managed. In addition, vast design spaces can be searched while solutions are intelligently modified, their performance evaluated, and their results aggregated into compatible sets for design decisions.

The EDM is composed of several design states as well as design evolving processes. A design state describes a design at a particular point in time and maps the system object to the

system's requirements and identifies its relation to the context in which the system will operate. A design evolving process involves many sub-processes which include formulation, decomposition, modeling, and integration. These sub-processes are not always carried out in a sequential manner, but rather a continuous move back and forth to previous and subsequent stages is expected.

The resulting design model is described as evolutionary model that moves a system's design from simple abstract states to more complex and detailed states throughout its evolution.

The thesis will be loosely partition into eight chapters the contents of which intersect necessarily. The chapter descriptions follow:

1. Introduction. This will provide a simple background on design science, systems theory, and computational modeling and their influence on the design process.

2. System Modeling Methodologies: This section will present some of the methods used for modeling systems including both Logical and Mathematical Models.

3. System Design and Designing: This chapter will represent the first step in understanding the EDM framework. It will discuss the difference between design as an object as well as design as a process.

4. The Design State Object: Here the design state will be described and the different sets within the design state will be investigated. Theses which include: the requirements, context and system sets.

5. The Design Evolving Process: In this chapter the design evolving process will be presented and its sub-processes which involve Formulation, Decomposition, Modeling and Integration

will be discussed.

6. Managing Complexity: while the EDM is constructed, several issues such as variation in design resolution, coupling as well as uncertainty should be considered. This chapter will discuss these issues.

7. Masdar City: A Case Study: This chapter will showcase an EDM experiment developed for Masdar city.

2. System Modeling Methodologies

In general, a *model* is an imitation or approximate representation of a system or of complex functions (Papalambros and Wilde, 2000). It is a simplified or abstract view of the complex reality using a physical, logical or mathematical representation of the system of entities, phenomena, or processes. It may focus on specific views, thus facilitating the understanding and analysis of complex problems through decomposition.

The type of model used in a design context is highly dependent on and driven by design intent. Design intents can be categorized into two classes: Logical design intents and mathematical design intents. Mathematical design intents can be identified and their performances can be measured numerically, whereas logical design intents represent logical structures and relations that do not necessarily have numerical values.

2.1 Logical Modeling Methods

Logical modeling methods can help in defining the system's architecture. It can describe it at different degrees of abstraction, and can demonstrate how various design activities are going to be connected together through compatible interfaces.

Logical modeling would typically take place before mathematical modeling and software programming to avoid major reprogramming later on. It basically promotes the interaction among the system architects, design specialists and

other project members, as well as allowing the visualization of control flow and data. Within

Several logical modeling methods have been designed for general-purpose systems, each with its own semantics and notation in the following sections I will discuss three of these methods: UML, SysML, and OPM.

2.1.1 Unified Modeling Language (UML)

UML (or Unified Modeling Language) is a general-purpose, standardized visual specification modeling language. It uses graphical diagrammatic representation to create an abstract model of a system, enabling software developers to model computer applications. This model is referred to as a UML model.

UML is programming-language independent and platform-independent. Its tools are used at length in J2EE and .NET shops. It has thus enabled software developers to focus more on design and architecture due to its common design language.

UML models are different from the represented set of diagrams of a system. A diagram is a partial graphical representation of the model. The model at the same time contains written use cases which act as documentation that drives the model elements and diagrams.

There are three main processes involved in UML models: visualizing, constructing, and documenting. Visualizing involves using diagrams for communicating the model as an idea into an expression in the form of diagrams. Constructing uses these visual illustrations in a prescriptive manner to build the system. Documenting involves using models and diagrams to capture knowledge of the requirements and system throughout the process.

UML Views

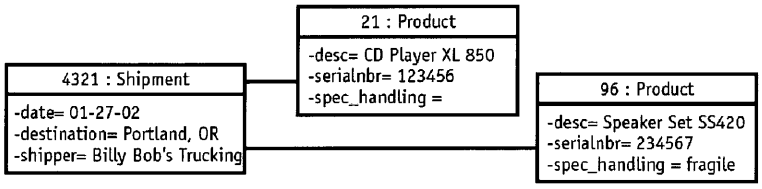
UML defines thirteen types of diagrams which represent three different views of a system model. Six diagram types represent static application structure; three represent general types of behavior; and four represent different aspects of interaction. In the following sections I will demonstrate some of these views and diagrams.

Static structural view

This view emphasizes the static structure of a system, meaning what must exist in the modeled system. This is done by using objects, attributes, operations, and relationships. Structure diagrams include the following diagrams: class diagrams, object diagrams, component diagrams, composite structure diagrams, package diagrams, and deployment diagrams (Figure 2.1).

Figure 2.1:

A complete class diagram (Pender , 2002).

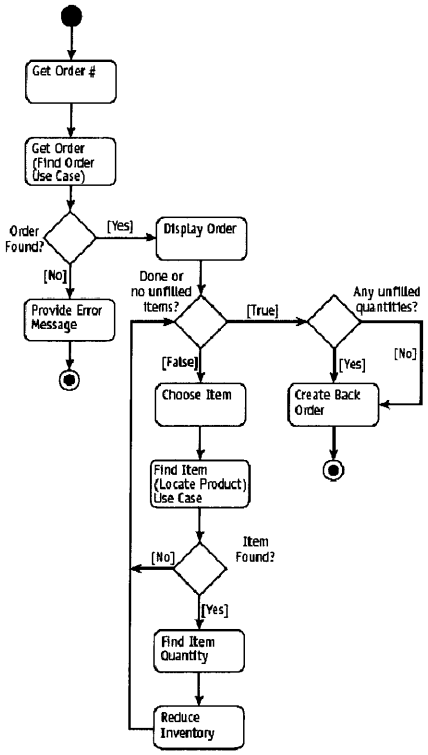


Behavior view

This view focuses on the dynamic behavior within a system, including changes to the internal states of objects. It also stresses on the collaborative activities and decisions among objects, describing what must happen in the modeled system. Behavior diagrams are primarily flowcharts and DFDs that are used to acquire the general flow of the code. They include the following diagrams: use case diagrams, activity diagrams, and state machine diagrams (Figure 2.2).

Figure 2.2:

Activity diagram with 3 swimlanes (Pender,2002).



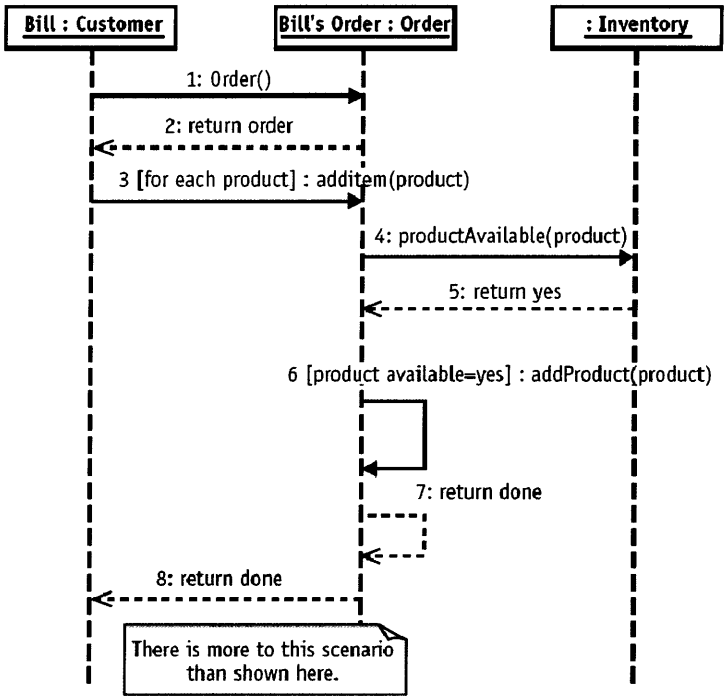
Interactions View

The Interaction diagram is a subset of behavior diagrams. It focuses however on the flow of data and control among the objects in the modeled system. Interaction diagrams include the following diagrams: sequence diagrams, communication diagrams, timing diagrams, and interaction overview diagrams.

In general, there is no restriction as to the appearance of all UML components on any types of UML diagrams. In terms of notation, usually the presence of a comment or note is allowed in a UML diagram, so that intent, usage, or constraints can be expressed and explained clearly. This is traced back to the conventional notation system used in engineering drawings (figure 2.3).

Figure 2.3:

A sample sequence diagram (Pender,2002).



2.1.2 Systems Modeling Language (SysML)

OMG SysML™ is a general-purpose graphical modeling language characterized by having computer-sensible semantics (OMG, 2007a). The main purpose of this language is the identification, analysis, design, and verification of complex systems. In a way, SysML adapts UML™, which is primarily used for modeling software-intensive systems, for the purpose of systems engineering applications. Similar to the UML approach in unifying modeling languages in the software industry, SysML reuses a subset of UML 2 to unify the wide range of modeling languages, tools and techniques currently in use by systems engineers.

The history of SysML goes back to 2001 when the International Council on Systems Engineering's (INCOSE) Model Driven Systems Design workgroup decided to customize UML for systems engineering applications. Two main bodies, the

INCOSE and the Object Management Group (OMG) (which maintains UML specification), collaborated as a result and jointly developed with the assistance of other groups the specifications for the SysML in March 2003 (OMG, 2007a).

SysML uses UML 2.0 and its extensions as its basic foundation. Therefore both systems engineers using SysML and software engineers who model using UML 2 can collaborate effortlessly on models of software-intensive systems. This enhanced communication among participants in the systems development process advances interoperability among modeling tools. It is most likely that SysML will be customized to model domain-specific applications, such as automotive, aerospace, communications, and information systems.

Figure 2.4:

SysML diagram taxonomy (OMG,2007b)

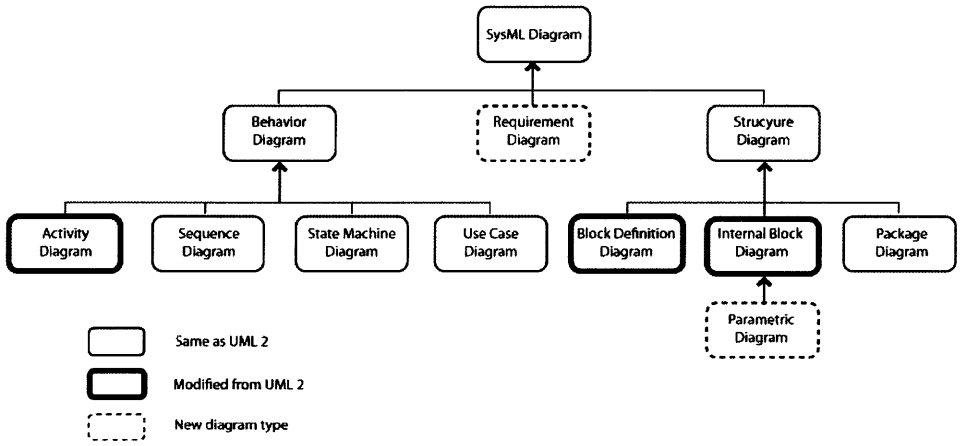


Figure 2.4 illustrates the SysML diagram taxonomy, representing the concrete notation for the diagrams, together with the corresponding specification of the UML extensions. Compared to UML, SysML is a smaller language, both in diagram types and total constructs, as it reduces many of UML's software-centric constructs. This makes it an easier language to learn and apply, and much more flexible and expressive.

SysML, like UML, supports allocation tables, a tabular format

that is dynamically derived from allocation relationships. While UML provides only limited support for tabular notations, SysML is characterized by flexible allocation tables that support requirement, functional and structural allocation. SysML constructs for model management extend UML capabilities and support models, views, and viewpoints.

SysML implements a total of nine diagram types, seven of which belong to the original thirteen UML 2.0 diagrams. It adds two other diagram types: requirements, used for requirements management; and parametric diagrams, used for performance and quantitative analysis.

A requirement specifies a condition that should be met. A requirement may specify a function that a system must execute or a performance specification a system must achieve. The requirements diagram can depict the requirements in graphical, tabular, or tree structure format. Other diagrams can also have requirements appear on them to show their relationship to other modeling elements. Modeling constructs are supplied in SysML to represent text-based requirements and their relation to other modeling elements. The requirements modeling constructs were developed to bridge between traditional requirements management tools and the other SysML models (OMG, 2007b).

Parametrics primarily supports engineering analysis of critical system parameters, well known as a crucial aspect of systems engineering. This analysis includes the evaluation of performance, reliability, and physical characteristics (OMG, 2007b). Parametrics addresses the gap in previous modeling languages such as UML, IDEF, and behavior diagrams. It also provides a mechanism that deals with problems in non-standardized engineering analysis models. Previous non-standardized engineering analysis models lack the integration and synchronization with system architectural models, which specify the behavioral and structural aspects of a system, due

to the complexity and diversity of engineering analysis models. Parametrics integrates engineering analysis models with system requirements and design models for behavior and structure. It is also represents constraints in order to capture other types of knowledge beyond engineering analysis.

With these augmentations, SysML can model many systems, including hardware, software, information, processes, personnel, and facilities. Principal terminology in SysML parametrics includes the following terms:

- *Constraints* are similar to equations. This is useful in most engineering problems. A *constraint block* defines this equation in a way that makes it reusable. A *constraint property* is a specific instance of usage of a generic constraint block (for example, supporting engineering analysis of a particular design).
- *Parameters* represent variables of an equation or a constraint.
- *Value properties* represent any measurable attributes of a system architectural model or its components that are subject to analysis (e.g. mass). Through binding, the generic equations are linked to the value properties that specify the system and its components. Thus, the value properties are said to be bound to the parameters of a constraint.

2.1.3 Object-Process Methodology (OPM)

Object-Process Methodology (OPM) is considered a comprehensive approach to soft modeling of complex systems enhancement, evolution, and lifecycle support. It comprises a combination of structural, functional and behavioral features of a system in a single unified view.

OPM consists of three entities: object, process and state. Objects are things that exist, while processes are things that

affect objects. States are situations at which objects can be. Processes transform objects in one of three ways: generating, consuming or affecting them. The effect a process has on an object is manifesting through a change in the object's state. There are two types of links that OPM uses to connect entities with each other: structural links and procedural links. Structural links express persistent, long-term relations among objects or among processes in the system, while procedural links express the behavior of the system. The two types of links, structural and procedural, are expressed in the same diagram to provide an image of the system.

OPM is expressed bi-modally through graphical and textual representations. The Object-Process Diagram (OPD) is a graphical representation of a systems model while Object-Process Language (OPL) is a textual representation.

OPM features a concise set of symbols that form a language for expressing the system's building blocks and how they relate to each other both structurally and behaviorally. Built-in refinement-abstraction mechanisms are among OPM features that handle model complexities.

Building Blocks

Building blocks in OPM are: objects, processes, and states. *Objects* are what a system or product is (physical or informatical), and they may have states, while *processes* are things that transform objects. At any particular point in time, an object can be exactly in one *state*, and object states are changed through processes (Reinhartz-Berger1 and Dori, 2004).

Processes can transform an object in three different ways: by creating it, by destroying it, or by affecting it in some way. When a process affects an object, it changes the state of that object. Procedural links within the OPD express this in graphics (Grobshtein and Dori, 2008).

Table 2.1:

The Building Blocks in OPM (Dori, 2002)

The Building Blocks

	Visual Representation	Textual Form	Definition	Description
Entities	**Object**	Nouns; capitalized first letter in every word; if ending with "ing", "Object" is place as a suffix	An ***object*** *is a thing that has the potential of stable, unconditional physical or mental existence.*	Static things. Can be change only by processes.
	Process(ing)	Nouns in gerund form; capitalized first letter in every word; if not ending with "ing", "Process" is place as a suffix	*A* ***process*** *is a pattern of transformation that an object undergoes.*	Dynamic things. Are recognizable by the changes they cause to objects.
	Object **state**	Nouns, adjective or adverbs: non- capitalized	*A state is a situation an object can be at.*	States describe objects. They are attributes of objects. Process can change an object's state.

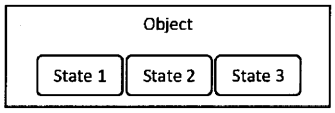
There are certain symbols that refer to entities these are rectangles and ellipses (Table 2.1). Rectangles refer to objects, while ellipses refer to processes. The symbol of state is rounded rectangle (Table 2.2).

Table 2.2:

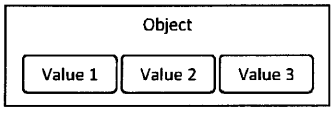
State diagrams in OPM (Dori, 2002)

States

State sentences and images



Object can be state 1, state 2, or state 3.



Values of Object are Value 1, Value 2, or Value 3.

The name of the object, process or state is recorded inside the corresponding symbol. To reduce the effort needed to read and understand objects and processes, OPM uses a naming convention in which the first letter in object and process names is always capitalized. In addition, to differentiate objects from processes, process names end with the suffix ing, indicating that they are active. States start with a lower-case letter (Dori, 2002).

Links

OPM has two types of links that connect entities with each other: procedural links and structural links. Structural links relate objects to other objects and processes to other processes and can express static relations between pairs of entities. However, structural links are not used to relate objects to processes. Procedural links, on the other hand, are links used to connect entities to describe the behavior of a system. They link an object to a process and can indicate a change in the state of the object.

Table 2.3:

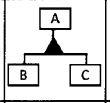
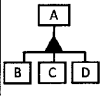
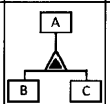
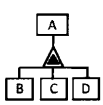
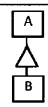
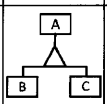
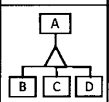
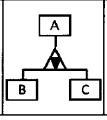
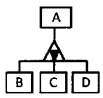
Structural links (Dori, 2002)

The Four Fundamental Structural Relations

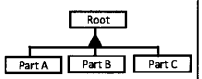
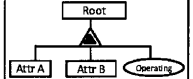
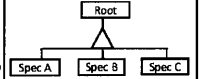
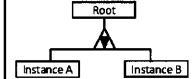
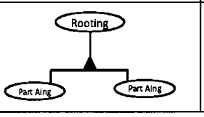
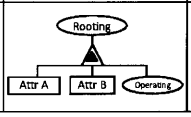
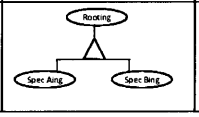
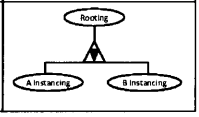
Shorthand Name	Aggregation	Exhibition	Generalization	Instantiation
Symbol				
Meaning	Relates a whole to its parts	Relates an exhibitor to its attributes	Relates a general thing to its specializations	Relates a class of things to its instances

A fundamental structural relation can have many descendants.

The different OPL sentences and OPD pictures are listed below.

Structural Relation Name and Shorthand Name	Number of Descendants						Description
	One		Two		Three or more		
	OPD	OPL	OPD	OPL	OPD	OPL	
Aggregation-Participation		A consist of B.		A consists of B and C.		A consists of B, C and D.	B, C and D are part of the whole A.
Exhibition-Characterization		A exhibit s B.		A exhibits B and C.		A exhibits B, C and D.	B, C and D are attributes of A. If B is process, it is an operation of A.
Generalization-Specialization		B is an A.		B and C are As.		B, C and D are As.	B, C and D are types of A.
Classification-**Instantiation**		B is an instanc e of A.		B and C are instances of A.		B, C and D are instances of A.	B, C and D are unique objects of the class A.

The four fundamental relations are also applicable to process. Only exhibition can link objects with processes. Instantiation cannot generate a hierarchy while the other three can. Any number of things can be linked to the root.

	Aggregation	Exhibition	Generalization	Instantiation
Object				
Process				

The main procedural links are the result link, the input link and the output link. All three types connect processes to objects or states. While graphically all three look alike, their distinction is inferable from their context, i.e., the combination of their source and destination.

This demonstrates the context sensitivity of the graphic symbols in an OPD. Depending on the link's context in the OPD the same link can have more than one meaning. Therefore, the same link may serve a few related purposes. This context sensitivity enables the small set of symbols in OPM to express rich semantics (Dori, 2002).

The four main structural relations are aggregation-participation, generalization-specialization, exhibition-characterization, and classification instantiation. General tagged structural links provide additional "user-defined" links with specified semantics (table 2.4), similar to association links in SysML block diagrams (Grobshtein and Dori, 2008).

Table 2.4:

Tagged structural links (Dori, 2002)

Tagged Structural Links

Generally used between objects, but may also be used between processes.
Cannot be used to link an object to a process.

Link Name	Object Process Diagram (OPD) Symbol	OPL SENTENCE	Description
Tagged	R Objects — refers to → S Object	**R Object refers to S Object.**	Relation from source object to destination object; relation name is entered by architect, and is recorded along link.
(Null)	R Objects → S Object	**R Object** relates to **S Object.**	Relation from source object to destination object with no tag.
Bi-directional Tagged	R Objects — precedes / follows → S Objects	**R Object precedes S Object. S Object follows R Object.**	Relation between two objects; relation names are entered by architect, and are recorded along link.
(Null) Bi-directional	R Objects ↔ S Objects	**R Object** and **S Object** are related.	Relation between two objects with no tag.

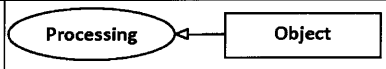
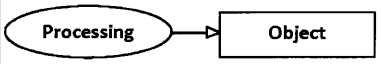
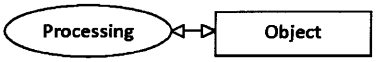
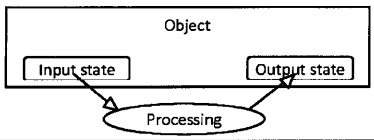
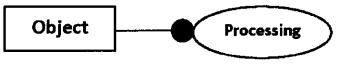
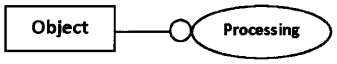
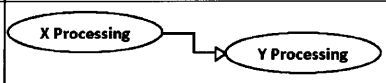
Other procedural links include enabling and event links. An enabling (agent or instrument) link expresses the need for an object to be present in order for the enabled process to occur. The enabled process does not transform the enabling object. An event link connects a triggering entity (object, process, or state) with a process that it invokes (table 2.5).

Table 2.5:

Procedural links (Dori, 2002)

Procedural Links

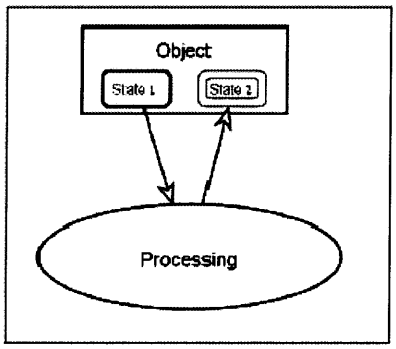
These links are generally used between an object and a process. They cannot be used to links objects together.

Link Name	OPD Symbol	OPL Sentence	Description
Consumption		**Processing** consumes **Object.**	Process uses object up entirely during its occurrence.
Result		**Processing** yields **Object.**	Process creates an entirely new object during its occurrence.
Effect		**Processing** affects **Object.**	Process changes the states of the object in an unspecified manner.
Input and Output		**Processing** changes **Object** from **input state** to **output state.**	The object is at input state prior to the process occurrence, and at output state as a result of its occurrence.
Agent		**Object** handles **Processing.**	Object is a human that is not changed by the process; process needs the agent object in order to occur.
Instrument		**Processing** requires **Object.**	Object is a non-human that is not changed by the process; process needs the instrument object in order to occur.
Invocation		**X Processing** invokes **Y Processing.**	First process directly stars up a second process, without an intermediate object.

In OPM there are three ways that explain a system's behavior: processes can transform objects, objects can enable processes, and objects can trigger events that invoke processes. Figure 2.5 shows pair of input and output links. It is expressed in OPL as "Processing changes Object from State 1 to State 2."

Figure 2.5:

Processing changes Object from State 1 to State 2 (Dori, 2002)



Complexity Management

A system can be presented as a set of inter-related, hierarchically organized OPDs that show portions of the system at various levels of detail. The root diagram in the model is the System Diagram (SD). It displays the most abstract view of the system. The SD typically shows a single process as the main function of the system as well as the most significant objects that enable it or are transformed by it. The more distance there is between the root and the OPD, the more details there is. Apart from for the SD, each OPD is obtained by refinement-either by zooming-in or unfolding- of an entity in its ancestor OPD. The security of the context, at any detail level, of an entity is ensured by the abstraction-refinement mechanism and the "big picture" is maintained at all times. A new entity can be presented in any OPD as a refineable of an entity at a higher abstraction level. Hence, copies of an entity can occur in other diagrams. The irrelevant or unnecessary details should not be shown in the context of a particular diagram. The three refinement/abstraction mechanisms are (Grobshtein and Dori, 2008):

1. The unfolding and/or folding that is used for refining/abstracting the structural hierarchy of an entity and can be applied by default to an object.
2. In-zooming and/or out-zooming which exposes and/or

hides the inner details of an entity within its frame and is applied mainly to processes.

3. State expressing and/or suppressing which exposes and/or hides the states of an object.

Reinhartz-Berger1 and Dori (2004) argue that the goal of complexity management in OPM is to balance the tradeoff between two conflicting needs: completeness and clarity. Completeness requires that the system details be as specific as possible, while the need for clarity imposes an upper limit on the level of complexity and does not allow for an OPD and its corresponding OPL paragraph to be cluttered or overloaded with entities and links.

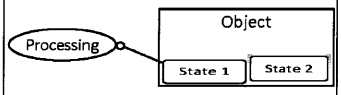
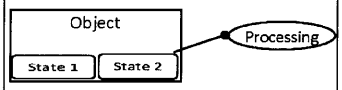
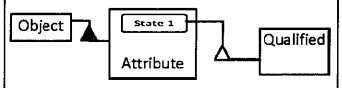
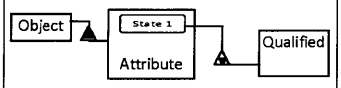
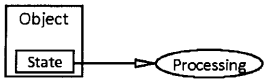
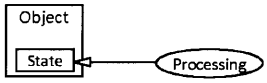
Bi-Modal Representation

OPM is expressed bi-modally through graphical and textual representations (table 2.6). The graphic representation is expressed through a set of Object-Process Diagrams (OPDs) which is the graphical, visual OPM formalism. The textual representation is expressed by the use of Object-Process Language (OPL). OPL specifies the same OPM model in a subset of English, enabling direct mapping between the graphic and the textual representations.

Each OPM element is denoted in an OPD by a symbol, and the OPD syntax specifies ways by which entities can be linked. OPDs consist of both the entities of the model (objects, processes, and states) and links and relations among them, in addition to data to preserve the graphical demonstration of the model elements (size, location, etc.).

OPL is a dual-purpose language, oriented towards humans as well as machines. Every OPD creation is expressed by a semantically comparable sentence or phrase of OPL that is automatically generated by an OPM-supporting modeling tool, such as OPCAT (Dori et al. 2003).

State-related Links

Link Name	OPD Symbol	OPL Sentence	Description
Condition		**Processing** occurs if **Object** is **state 1**	Object is an instrument. It must be at a specific state in order for the process to occur.
Agent Condition		**Object** must be at **state 2** for **Processing** to occur.	Object is an agent. It must be at a specific state in order for the process to occur.
Qualification		**Qualified Object** is an **Object,** the **attribute** of which is **state 1.**	Qualified Object is a type of Object. It must be at a particular state of Object's Attribute.
Instance Qualification		**Qualified Object** is an instance of an **object,** the **Attribute** of which is **state 1.**	Qualified Object is an instance of class Object. It must be at a particular state of Object's Attribute.
State Specified Consumption		**Processing** consumes **state Object**	Process consumes object only if it is at a certain state.
State Specified Result		**Processing** yields **state Objects.**	Process creates objects at a certain state.

Boolean Objects

Specialized informatical objects. Boolean objects are questions, and they always have two states (the answers): yes and no.

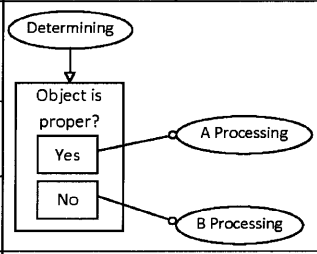
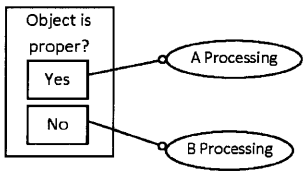
Link Type	OPD Symbols	OPL Sentence	Description
Determination (a Result Link)		**Determining** determines whether **Object** is **proper.**	Process yields a Boolean object that poses a "yes or no" question. The process then determines the answer.
Condition link		**A Processing** occurs if **Object** is **proper.**	If the answer is "yes", a certain process occurs. If the answer is "no", a different process occurs.
Negative condition link		**B Processing** occurs if **Object** is **proper.**	
Both condition links		**A Processing** occurs if Object is **proper,** otherwise **B Processing** occurs.	Compound sentence; if the answer is "yes", a certain process occurs, otherwise a different process occurs.

Table 2.6:

State related links (Dori, 2002)

OPL sentences contain non-reserved words as well as reserved words. The non-reserved words are domain-specific words which a system designer uses to describe a specific system. On the other hand the reserved words link the non-reserved words

to create a natural language sentence. Dori (2002) believes that OPL sentences are more understandable than a script of a computer programming language.

Dori (2002) claims that Object-Process Diagrams (OPDs) and Object-Process Language (OPL) provide harmonized, crisp representations of the system. Both processes complement each other and appeal to the two sides of the brain at once.

OPCAT

OPCAT (Dori et al. 2003) is a software product that sustains the development and lifecycle management OPM-based system by applying the hierarchical, bimodal expression of the OPM model. System complexity management is supported by the OPCAT platform, including interconnections and traceability to the model entities. Features such as animated simulation of the model, code generation, and automatic document generation are included as well.

2.2 Mathematical Modeling Methods

Within a design problem a mathematical model consists of a set of quantitative and logical statements that represent relevant features of a specific system in terms of mathematical concepts, symbols and language including variables, parameters, and relationships such as equations and inequalities (Jacoby and Kowalik, 1980; Maki and Thompson, 2006).

A mathematical model becomes a computational model as soon as its associated equations are coded into a computer program where it can be studied numerically and graphically (Maki and Thompson, 2006). Simulation, as one of the applications that involves intense computation, deals primarily with the process of designing a model of a system and conducting experiments on that model. The relationships in the

model are manipulated to observe how the model reacts, and how the system would eventually react accordingly if the mathematical model were valid (Averill, 2006). This allows for testing hypotheses at a much lower cost than actually performing that activity in reality.

Types of Mathematical Models in Design

Coyne *et al.* (1990) have written that "In modeling design we do not attempt to say what design is or how human designers do what they do, but rather provide models by which we can explain and perhaps even replicate certain aspect of design behavior."

To model a design mathematically we must be able to define it fully. Designers and engineers regularly utilize mathematical models to perform typical design activities. These activities include generating one or more physical configurations, known as synthesis. They also include studying the performance and behavior of these configurations through engineering and science which is known as analysis. Designers and engineers then have to make design decisions about the results, which is known as evaluation. Finally they have to devise mechanisms for searching for the best alternative(s), which is known as optimization.

2.2.1 Synthesis Models

Many researchers have attempted to formalize synthesis processes of design aiming at achieving shorter design cycles and more robust solutions. With the introduction of computers, formal synthesis methodologies and structured algorithmic descriptions were implemented computationally. This computational approach has the advantage of managing and tackling problems that are not open to solution by humans.

Synthesis models require a type of representation, specifically

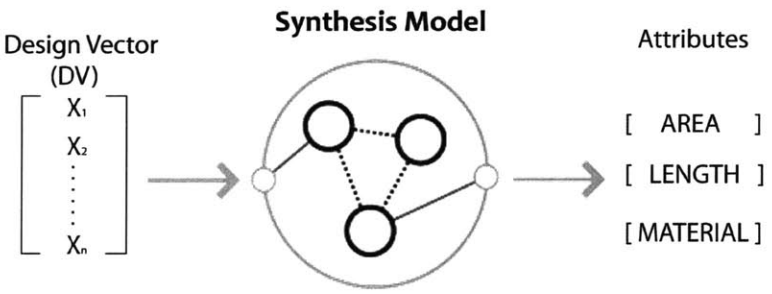
for the geometric attributes of the system. The representation of a system in a synthesis model requires setting up clear definitions of their composing elements, and the operations that can help implement them. Modeling systems is typically partial and so are the representations expressing them.

Geometric representation of artifacts in a computer environment can be constructed in the form of wireframes, surface or solid representations. Wireframe representations can be thought of as a set of curves that describe space discontinuities. Surface representations are built off of wireframes. They describe two dimensional spaces. Wireframe and surface representations are ambiguous. They do not provide information on what is inside or outside, what is filled or empty. Solid models offer a better depiction of physical artifacts for they are un-ambiguous. They provide information on closure (or water-tightness), boundaries, inside and outside, and well-formedness allowing for automated manipulation and testing (Kalay, 1989).

The input to the synthesis model is a design vector. Both the design vector and the structure of the synthesis model affect the nature of the solution space. A synthesis model is expected to output a solution and certain attributes that then become the input to the analysis models (figure 2.6).

Figure 2.6:

Expected input and output of the synthesis model.



There are well established formalisms that are intended to capture design intent and relations. These are known as formal grammars. The term formal grammar originated from

Chomsky's work on linguistics in 1956 (Chomsky, 2002). A formal grammar is a set of instructions for sequencing a set of symbols to form valid words. The set of all words generated by a grammar formulates a language. Building vocabulary is similar to mathematical modeling for it entails describing sequences of symbols and operations. The study of formal grammar properties in mathematics is called formal language theory.

Formal grammars, do not only provide instructions to synthesize (generate) strings (concatenations of symbols) in a language, but also determine if a given string belongs to a language through analyzing its internal structure.

Grammars demonstrate a robust structure for processing information as they can pack logic of a whole language, and generate its entire set of solutions. A synthesis grammar language is typically expressed as:

$$G = \{V, R, S\}$$

The grammar G is a model that includes: a set of vocabulary V, a set of rules R and a set of initial states S. The set of vocabulary V is expressed via a certain representation. The notion of symbol manipulation in formal grammars indicates that they deal with clearly defined vocabulary which is not limited to alphabets. In general terms, a symbol in a vocabulary is a representation of an element.

The rules set R includes conditional constructs (IF-THEN, DO UNTILL, etc) that fire replacement algorithms. Formal grammars sequences (instructions) manipulate symbols by a process of replacement and therefore can be treated within a computer program as production systems. Replacement rules are typically expressed in the form of $X \rightarrow Y$, which means IF X is found, THEN it should be replaced by Y. Replacement rules can be sequenced in many fashions such as stochastic, procedural approaching a certain state, or recursive where a rule keeps

invoking itself until a certain condition is achieved (Mitchell, 1990).

Synthesis grammars also require an input which is a set of initial states S. The initial states set includes the left side of design rules. Initial states define the nuclei that can be used to initiate a solution and resemble the left side of design rules.

Formalisms that will be discussed in this section include: Lindenmayer Systems, Graph grammars, Cellular Automata, and Shape grammars.

L-Systems

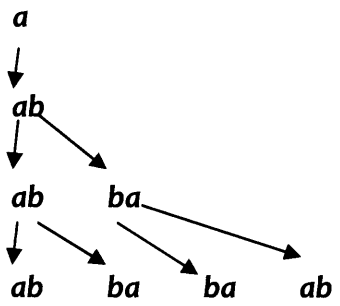
Aristid Lindenmayer introduced L-systems in 1968 as a method to describe and simulate growth of multi-cellular organisms, typically plants. L-systems perform two main tasks: representing (packaging) information in symbols and interpreting those symbols as growth patterns. The elements (symbols) that exist in an L-system are called axioms. The initial string is a composite of axioms.

L-systems operate by replacing symbols with one another based on replacement rules. Rules rewrite input strings sequentially. Each step of rules execution represents a generation. Expressing generations (outcome) in an L-system is analogical to providing instructions of how a solution unfolds as opposed to providing blueprints that describe every element in the final solution (Hemberg, 2001).

Furthermore, L-systems are implemented in parallel as opposed to sequential (Hemberg, 2001). The nature of rule implementation in an L-system makes it hard to hand-make the system produce a specific result. A typical rule representation in an L-system is shown below. The following rules replace "a" with "ab", and "b" with "ba".

$$a \rightarrow a\,b$$
$$b \rightarrow b\,a$$

if started with the symbol *a*, produces the following strings,



Rules application in an L-system can be guided by parameters that determine which one to execute. This type is known as Parametric L-systems. In a Parametric L-system, rules consist of three components: the predecessor, the condition and the successor. For example, a production with predecessor $A(n_0, n_1)$, condition $n1 > 5$

and successor $B(n_1 + 1)cD(n^1 + 0.5, n_0 - 2)$ is written as:

$$A(n_0, n_1): n_1 > 5 \rightarrow B(n_1 + 1)cD(n^1 + 0.5, n_0 - 2)$$

A production matches a module in a parametric word *iff* the letter in the module and the letter in the production predecessor are the same, the number of actual parameters in the module is equal to the number of formal parameters in the production predecessor, and the condition evaluates to true if the actual parameter values are substituted for the formal parameters in the production (Hornby and Pollack, 2001).

For example , the PL system,

$$a(n) : (n > 1) \rightarrow a(n-1)b(n)$$
$$a(n) : (n \leq 1) \rightarrow a(0)$$
$$b(n) : (n > 2) \rightarrow b\left(\frac{n}{2}\right)a(n-1)$$
$$b(n) : (n \leq 2) \rightarrow b(0)$$

When started with (4) , produces the following sequence of strings,

$a(0)$
$a(3)\ b(4)$
$a(2)b(3)b(2)a(3)$
$a(1)b(2)b(1.5)a(2)b(0)b(1.5)a(2)$
$a(0)b(0)b(0)a(1)b(2)b(0)a(1)b(2)b(1.5)a(2)$
$a(0)b(0)b(0)a(0)b(0)b(0)a(0)b(0)b(0)a(1)\ b(2)$
$a(0)b(0)b(0)a(0)b(0)b(0)a(0)b(0)b(0)a(0)\ b(0)$

Graph Grammars

A graph *G(N,E)* consists of a set of nodes *N* and a set of relations $E \subseteq N \times N$ called edges, whereby the graph nodes as well as the edges have a label assigned to each. Nodes can include information such as attributes or constraints. Constraints define, or rather filter, the set of relationships between nodes by providing information on what can be considered as valid relationships (Alber, 2002).

Graph grammars are similar to other types of synthesis grammars in that they are composed of an initial vocabulary V, and a set of rules R, and initial states S. The vocabulary consists of labeled and attributed nodes. The initial statess_i is any structured combination of the vocabulary elements. A specific axiom or initial state together with an ordered set of rules out of define a production system and corresponds to one specific graph which can be constructed following this program.

During the execution of a production system the initial graph s_i is modified by the graph rules thereby evolving in several stages and forming the graph *evolution sequence* $\{G_i^0, G_i^1, G_i^2, \ldots\ldots, G_i^n\}$ with$G_i^0 = s_i$.

Graph grammars deal with attributes and constraints dictating what links may be valid. They also deal with modifying, editing,

removing elements from the graph structure when inserting new nodes.

Cellular Automata

A cellular automaton is a collection of cells organized in orthogonal grids, each with a finite set of states. This collection of cells evolves over discrete time steps based on the notion of neighborhoods.

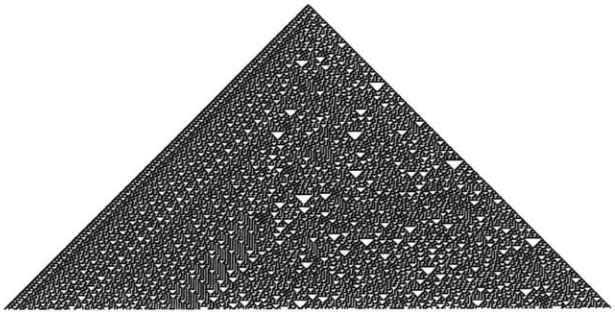
A cell changes its state based on its current state and its neighboring cells states following rules. A solution in CA is generated once every cell in the collection runs the embedded rules. CA are sequential, meaning the behavior (state change) in each cell depends on how its neighbors behave. Unlike L-systems were rules are applied in parallel.

The simplest type of cellular automata is linear, known as elementary CA. Each cell in elementary cellular automata has two states, black or white. To calculate the number of possible neighborhoods of three cells, we raise the number of states to the number of cells in a neighborhood, so 2 ^ 3 = 8 types of neighborhoods. To calculate the number of possible combinations of 8 neighborhoods with two states, we raise the number of states to the number of neighborhoods, 2 ^8 = 256 possible combinations of neighborhoods of 3 cells each and 2 states per cell. The total number of possible neighborhoods is known as possible CA rules.

Rules in elementary CA are represented as arrays of back (1) and white (0) unites. They are labeled by calculating the locations of black cells on a binary scale of 128 64 32 16 8 4 2 1. In these location, any black cells. So the representation for a rule 30 is as follows

Figure 2.7:

CA rule 30



00011110 → (0*128) + (0 * 64) + (0*32) + (16*1) + (8*1) + (4*1) + (2*1) + (0*1) = 0+0+0+16+8+4+2+0 = 30.

Figure 2.7 shows the result of rule 30.

Cellular automata are known to demonstrate four types of behavior: fixed point, periodic, chaotic and random. These types are defined based on the pattern of occurrence of certain behaviors over a defined time period.

Shape Grammars

Invented by Stiny and Gips in 1972 (Stiny and Gips, 1972), Shape grammars are a geometrical construct that express production algorithms and rules through basic geometric elements, points and lines. Shape grammar rules can be interpreted as replacement rules for they consist of a left side (initial shape), an arrow noting an operation, and a right side (the result). Shape grammar operations include addition, subtraction, intersection, and transformations. Transformations include: translation, reflection, rotation and scale.

In shape grammars, shapes are more of topological structures than geometric representations (Cagan, 2001). Topological elements do not intersect. They may exist in spaces of similar of higher dimensionality. The following table describes the algebra of shapes as Uij, where i represents the dimension of

the topological element, and j represents the space that accommodates it. For example the symbol U12 means it is a segment element that exists in a plane space.

U00	*U01*	*U02*	*U03*
	U11	*U12*	*U13*
		U22	*U23*
			U33

Like L-systems, Shape grammars can be also parameterized. A Parametric shape grammar is composed of the following tuple: (S, L, T, G, I). S is the expression of a Shape Grammar rule in the form (A→ B) which typically means: if shape A is found, it is to be replaced by shape B. L is the set of labels. Labels are notations added to the Shape Grammar rules. T is the set of geometric transformations that build into the Shape grammar rules. G is a set of functions that assign values to rules parameters. I is the set of initial shapes which Shape Grammar rules use as to start the calculation. Initial shapes are the left side of a Shape grammar rule (Kalay, 2004).

Shape grammar rules that combine various representations are known as parallel grammars such as combining description and shape rules in a grammar. While shape rules are applied to the evolving design geometric shapes, the corresponding description rules are applied to the evolving description. Thus, as the generation of the design evolves, the description of the design is constructed.

Recognition in Shape Grammars is based on the notion that shapes are non-atomic. They can be decomposed and recomposed freely at the discretion of the designer. Decomposition of elements in Shape Grammars is based on the notions of Embedding and Maximal Elements. Any element is considered a maximal element that includes all elements of similar topology but in smaller size. This notion of recognition in shape grammars make them virtually unlimited. As long as

the system is able to recognize an initial shape that a rule requires, the system will keep running. Emergence within this context is the ability to recognize shapes throughout a computation that were not explicitly defined (Duarte, 2001).

However shape grammars do not lend themselves well to computational implementation due to the complications associated with representing shapes numerically as well as the lack of current computational algorithms to recognize emergent shapes.

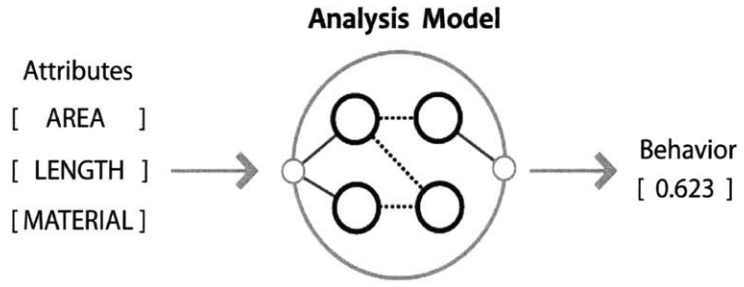
2.2.2 Analysis Models

Analysis models are developed according to principles of engineering science (Papalambros and Wilde, 2000). These models, which incorporate different analysis results, are constructed with the purpose of predicting the overall behavior of the design.

In an analysis model the inputs denote the specific attributes under which the behavior of that artifact is examined, while the output defines that behavior (figure 2.8). Those attributes are represented in terms of geometry, parameter values, boundaries, and initial conditions.

Figure 2.8:

Expected input and output of the analysis model.



In general, mathematical analysis models can be classified according to the type of model data, parameters and mathematical expressions into: qualitative or quantitative,

continuous or discrete, deterministic or stochastic, static or dynamic, linear or nonlinear, or any combination of these categories (Figure 2.9).

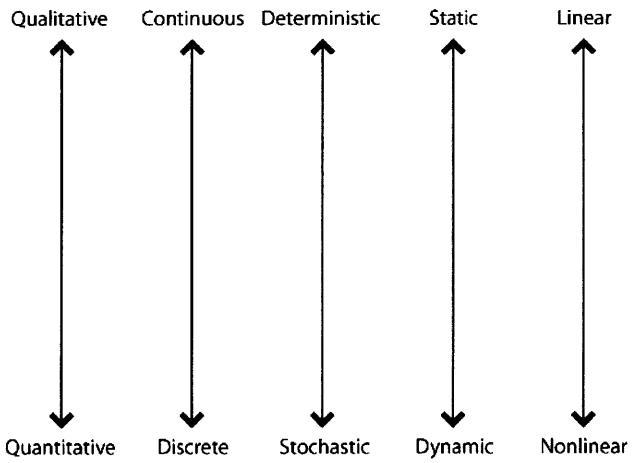
Analysis models can also be classified into empirical models, theoretical models, and reduced-order (or approximation) models.

Empirical models, which are typically low-fidelity models, are derived from observation and approximate data fitting rather than physics and first principles.

Theoretical models, on the other hand, are more physics-based and are derived using first-principle equations. They include both analytical and numerical models. Analytical models are mostly low-fidelity models whereas numerical models tend to be high-fidelity (high order) models that include models like Finite Element Analysis (FEA) and Computational Fluid Dynamics (CFD).

Figure 2.9:

Analysis models vary based on their mathematical nature.



Reduced-order or approximation models are surrogate models that provide simplified abstractions and calculations. They include response surface models, neural networks and Kriging

models. These models approximate the behavior of a design solution as closely as possible while maintaining low-fidelity, which is computationally cheaper.

High-fidelity simulations are generally considered better predictors of performance than low fidelity simulations, as they better resemble the artifact if administered correctly. However, the amount of fidelity necessary to guarantee good prediction is unknown. Also, there are some disadvantages to high-fidelity simulations that may render them less useful, such as trading off speed for accuracy. These types of simulations are not as quick and easy to construct as low-fidelity models.

Low-fidelity models are generally used to quickly demonstrate general system performance and abstract conceptual approaches in early design stages without providing much detail or requiring much investment in development. They are mostly used if some required data for the analysis model are not available, if the model cannot be easily quantified, or if a high-fidelity analysis is beyond the scope and accuracy level of the design description. Low-fidelity models also require a facilitator who knows and understands the domain in detail in order to illustrate or test the model.

During the model selection process, the designer must choose the best compromise between the demand for simplification and the necessity to clearly identify, describe and rate the targeted physical mechanism. A trade-off must be made between fidelity and analysis time and between simplicity and the accuracy of the model.

Analytical models are typically employed when the model and the relationships making it up are simple enough such that mathematical methods (e.g. algebra, calculus, or probability theory) can work with these relationships and quantities to obtain precise and explicit information regarding questions of interest (Averill, 2006). This information is known as an

analytical solution or a closed-form solution that can be simply arrived at with merely paper and pencil (Gershenfeld, 1998). This analytical solution allows for the prediction of system behavior through a set of initial conditions and parameters.

Analytical modeling is mostly done with analytic functions (Saff & Snider, 1993), and therefore the functions encountered are always assumed to be expanded in a power series. Analytical models are still considered very significant due to their power. It is almost always possible to deduce everything that needs to be known about a system using these models. This comes however at the expense of limited applicability, as many systems in the world are too complex to be described in this manner (Gershenfeld, 1998).

Although I stated above that analytical models are not computationally intensive, analytical solutions can occasionally be complex and call for immense computing resources. Obtaining a numerical solution for a situation where an analytical formula exists in theory, could be a very difficult task, such as the example of inverting a large nonsparse matrix (Averill, 2006).

Also, in many cases the closed form solution implied by analytical modeling is not usable for modeling experiments (Jacoby and Kowalik, 1980). If for example the model consists of a series that comprises many terms that need to be computed for solution accuracy purposes, the process of reformulating the problem as a numerical problem might be more economical. This reformulation takes the form of a sequence of consecutive approximations to the solution that are computed in an iterative manner. In these iterations, each approximation is "better" than its predecessor.

However, it is always preferred to study a mathematical model analytically rather than numerically if an analytical solution exists and is computationally efficient. Analytical models still

remain an important factor in some approximation techniques using computers. This extends to include numerical methods, where these methods can use bits and pieces of analytical solutions to render the numerical steps more effective. They also employ symbolic methods that can extend quantitative abilities and introduce important qualitative implications, such as in enhancing the methods to perform higher-order approximation theory (Gershenfeld, 1998).

Many real-world systems are too complex for analytical modeling or evaluation. It is clear then that not many differential equations can really be solved with precisely the same effort analytically. As we move farther from linearity, it becomes obvious that special techniques are required and enormous effort must be made to be able to write down a closed form or analytical solution (Gershenfeld, 1998). In the computer environment, differential equation models are typically reformulated and expressed in terms of difference equation approximations. Therefore the issue is reduced computationally to solving problems in the form of a set of algebraic equations (Jacoby and Kowalik, 1980).

According to Jacoby and Kowalik (1980), numerical solution methods for modeling problems basically have three main goals. These include computational efficiency, precision and error control, and solution convergence, where it is possible at some point to end the computation. They also add a few more points that could also be taken into account, such as the ability to solve altered or extended formulations, the availability of software that can successfully implement a specific method, and the conceptual clearness and ease of use of solution methods.

Several efficiency measures have been developed for numerical methods. These include the counts of arithmetic operations required to solve problems and the order of solution convergence in the case of iterative processes. One of

the most practical measures of efficiency, however, is the total computer resource required for solving the given problem, including time and storage space (Jacoby and Kowalik, 1980).

It is often hard to relate the latter measure to more theoretical properties of the model formulation and solution method. There are thus a group of mathematical techniques that help reduce the total computer resource needed for a solution. These include the decomposition of large-scale problems into smaller components in a semi-independent manner, using linearization if possible, data compression, and the process of reducing problems with unknown degrees of difficulty into well-known problems with which the model user is familiar and has relevant experience.

Accuracy of the numerical solution is another issue, which largely relies on the quality of data, the degree of model approximation, and the numerical properties of the solution method. Any one of these factors can nullify the solution results by itself. Recent developments in numerical analysis have made it easier for model users to comprehend many of the issues regarding error analysis and the conditioning of numerical problems and algorithms. These developments have enabled the understanding of the conditions under which any analysis mathematical model can be successful and useful.

The numerical results of the model should be interpretable and validated in the system space or else this information will remain uninterpretable in the analysis model space and would thus become unfamiliar to the model user. The mathematical modeling problem itself must be solvable in order to conduct the experiments correctly. This implies two conditions, existence and stability, meaning that the solution to the problem must exist theoretically and at the same time must always rely on the given side conditions. Any discontinuity must be accounted for appropriately.

Another condition for the success of the analysis model is uniqueness, where it should be understood beforehand whether the mathematical modeling problem allows for more than one solution or not. Whether or not the problem is well-conditioned is another important factor. This implies knowing whether small approximations in problem data result in small approximations in the final solution or not.

Finally, the feasibility of a computational process for numerical approximation is another important condition, where the model user must be allowed to keep the approximation error under control (Jacoby and Kowalik, 1980). Studying errors constitutes a very significant part of numerical analysis.

Errors can be introduced in the solution of the problem in a variety of ways. Round-off errors exist because it is impossible to represent all real numbers precisely on finite-state machines such as digital computers. Truncation errors exist after an iterative method is terminated and the approximate solution turns out to be different from the exact solution. Discretization errors also occur in the same manner, when the solution of the discrete problem does not match the solution of the continuous problem. As a general rule, an error generally propagates through the calculation once it is generated. If this propagation does not grow and accumulate in the input data and intermediate calculations causing a meaningless output, the algorithm is said to be numerically stable. This stability occurs only if the problem is well-conditioned, implying that the solution only changes by a small amount when the problem data is changed by a small amount. A well-conditioned problem does not necessarily guarantee the numerical stability of the algorithm, but an ill-conditioned problem definitely leads to error accumulation and consequently instability.

There have been several methods and algorithms developed for numerical models. These could be direct or iterative methods. Iterative methods are generally more common in

numerical analysis than direct methods.

In direct methods, the solution to a given problem is computed in a finite number of steps. The accurate answer can be provided through these methods if they are performed in infinite precision arithmetic. Finite precision is used in practice, and the end result represents an approximation of the true solution assuming stability. Examples of these methods include Gaussian elimination, the QR factorization method for solving systems of linear equations, as well as Cholesky and LU factorization (Trefethen and Bau, 1997).

Iterative methods are usually needed for large problems in computational matrix algebra. Unlike direct methods, iterative methods are not expected to be complete in a specific number of steps. They start from an initial guess to construct consecutive approximations that converge to the exact solution only in the limit. To determine when an accurate solution is found, a convergence criterion is specified. In general, even if iterative methods use infinite precision arithmetic, the solution would not be reached within a finite number of steps. Some examples include Newton's method, the bisection method, and Jacobi iteration (Trefethen and Bau, 1997).

Some methods, although direct in principle, are used as if they were not, such as GMRES and the conjugate gradient method. In these methods, the required number of steps to obtain an exact solution is large to the extent that approximations are accepted similar to the case of iterative methods.

Many methods have been developed for solving systems of linear equations. Standard direct methods that use matrix decomposition include Gaussian elimination, LU decomposition, Cholesky decomposition for symmetric and positive-definite matrix, and QR decomposition for non-square matrices. For large systems, iterative methods are preferred,

such as the Jacobi method, Gauss–Seidel method, the successive over-relaxation and conjugate gradient method (Trefethen and Bau, 1997). Root-finding algorithms and linearization are both techniques that are used for solving nonlinear equations. Newton's method is also used but when the function is differentiable and the derivative is known.

There are many other methods that are used to solve partial differential equations. Discretization is an approach that describes the process in which a continuous problem is substituted by a discrete problem whose solution is known to approximate that of the continuous problem. These methods face a major challenge that requires generating an equation that approximates the equation to be studied while being numerically stable. One of the methods used in this regard is the Finite Element Method, which is a good choice for solving partial differential equations.

2.2.3 Evaluation Models

Evaluation models aid the process of selecting good designs that constitute a compromise of several different requirements. This means that a design can be altered to create different alternatives with the ultimate goal being to choose the most desirable alternative. A decision has to be made once there is more than one alternative to choose from.

The model helps provide a clear explanation, prediction and a foundation for objective decision-making. The rational selection of an alternative requires a criterion which helps evaluate all alternatives and rank them according to best fit. The criterion used in such models is known as the objective of the model (Papalambros and Wilde, 2000). It is not unique, however, and its selection will be affected by a variety of factors. These include the design application, timing, point of view, the designers' own judgment, and the position of the individual in the hierarchy of the organization (Papalambros

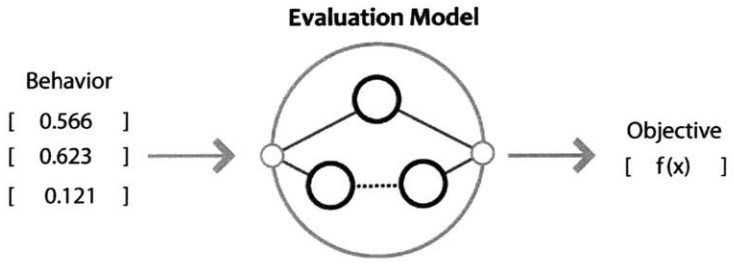
and Wilde, 2000).

Evaluation models are in essence decision-making tools. An evaluation model provides a quantitative assessment of the effects of design decisions on the system being considered. An evaluation model provides an objective evaluation as opposed to a subjective evaluation of system behavior.

In single objective design problems, the optimization and search direction can be well defined and a single solution, if it exists, could be found. However, as the design develops, more than one objective function will often be identified. These objectives may be competing and therefore trade-offs must be made. This implies that there is no single optimal solution but rather a whole set of possible solutions of equivalent quality (Abraham et al., 2005).

Figure 2.10:

Expected input and output of the analysis model.



Multiobjective optimization methods can be broadly classified into two categories: Decision making before search methods which are also known as Scalarization approaches, and Search before decision making methods which are also known as Pareto approaches.

In the first category of methods the designer decides how to aggregate different objectives into a single objective function before the actual search is performed (Gries, 2004). This requires the formation of a single objective function that contains contributions from the sub-objectives in vector *J*. The formation of the aggregate objective function requires that the preferences or weights between objectives are assigned

before the results of the optimization process are known. In this way, well-established single optimization methods can be applied (de Weck, 2004).

In the second category, the search for optimal solutions is performed with multiple objectives kept separate during the search. These Pareto methods typically use the concept of dominance to differentiate between inferior and non-inferior solutions. The result of the search is a set of Pareto-optimal solutions. Additional criteria or preferences can be applied after the search to find an optimal solution for a given problem. In this manner an unbiased search can be performed. In addition, a single search can serve several problem-specific decisions without the need to repeat the search (Gries, 2004). Therefore, the selection of a single- or a multi-objective search algorithm influences not only the point of time when design objectives are defined, but also influences the whole exploration process (Gries, 2004).

Table 2.7:

Different Scalarization and Pareto Methods (de Weck, 2004).

Scalaraization Methods (apriori preference expression)	**Pareto Methods** (a-posteriori preference expression)
• Weighted Sum Approach • Multiattribute Utility Analysis (MAUA) – Utility Theory. • Compromise Programming (Non-linear combinations). • Physical Programming, Goal Programming. • Lexicographic Approaches. • Acceptability Functions, Fuzzy Logic.	• Exploration and Pareto Filtering. • Multiobjective Genetic Algorithms (MOGA). • Weighted Sum Approach (with weight scanning). • Adaptive Weighted Sum method (AWS). • Normal Boundary Intersection (NBI). • Multiobjective Simulated Annealing (MOSA).

Decision Making before Search

In the decision making before search approach, multi-objectives are formulated into a scalar substitute problem that has a scalar objective and can be solved with the usual single

objective optimization methods. This method is called scalarization and is based on the assumptions that the designer preferences are known and assigned before searching the design space for design solutions.

The scalar objective has the form $f(x, p)$, where p is a vector of preference parameters that can be tuned to the designer's subjective preferences. The z objectives can be aggregated to express a utility, U, a dimensionless scalar quantity expressing the quality of a particular design.

$$\begin{aligned} &\max \{U(J_1, J_2, \dots, J_z)\} \\ &\text{s.t.} \quad J_i = f_i(x, p) \quad 1 \leq i \leq z \\ &\qquad x \in S, U \in R^+ \end{aligned}$$

Several scalarization methods have been developed (Table 2.7). The focus in the following will be on two of these methods namely the weighted-sum approach and the utility function approach.

Method of Weighted-Objectives

One of the most common and easiest to understand scalarization techniques is the weighted sum approach which is also known as the method of weighted-objectives. The scalar substitute objective is obtained by assigning subjective weights to each objective and summing up all objectives multiplied by their corresponding weight (Papalambros and Wilde, 2000).

The decision maker weights the different criteria according to their relative importance in determining the quality of a solution. This numerical treatment facilitates comparison among criteria that are not related (Kockler et al., 1990). Weighting should follow a logical breakdown.

This approach is characterized by one composite or utility function U declared by aggregating multiple objective

functions with individual weighting factors λj.

$$\begin{aligned} &\max \quad U\left(J(x,p)\right) \\ &\text{where} \quad U = \sum_{j=1}^{z} \lambda_j \frac{J_j}{sf_j} \text{ with } \lambda = [\,\lambda_1 \;\; \lambda_2 \;\; \ldots \;\; \lambda_z\,]^T \\ &\text{and} \quad \lambda \in R^z \;\; | \;\; \lambda_i > 0, \sum_{i=1}^{z} \lambda_i = 1 \\ &\text{and} \quad x \in S \end{aligned}$$

Formulated in this manner the objective *U* always forms a strictly convex combination of objectives. The individual objectives are typically normalized, and since the optima of the problem does not change if all weights are multiplied by a constant value, weights are chosen such that they add to unity and are themselves positive scalars (de Weck, 2004).

It is apparent that the preference of an objective can be changed by modifying the corresponding weighting factor which leads to another solution point. In the case of two equally scaled objectives:

$$U = \lambda J_1 + (1 - \lambda) J_2$$

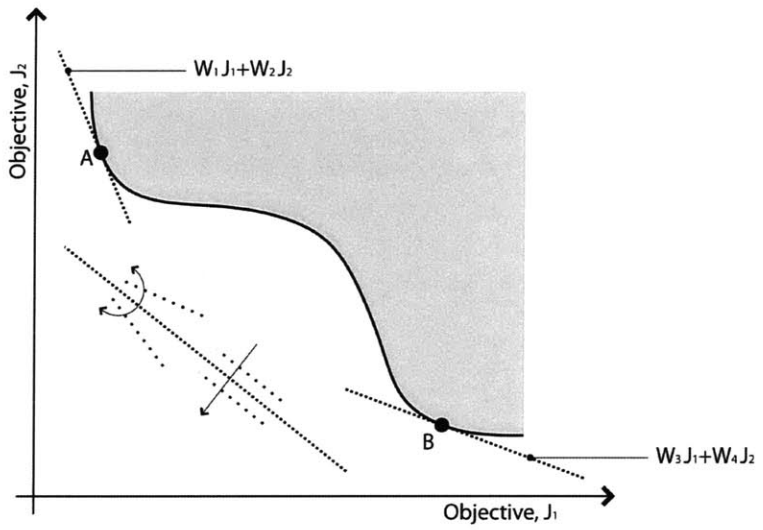
The ratio of the weights defines the constant slope of the line. Varying λ gradually in small incrementing steps exposes a set of optimal solutions as the weight is gradually shifted from one objective to another. This sequential variation of some weighting factors can be used to find as much trade-off solutions as possible.

This approach can be utilized to find the Pareto-front by obtaining different points on the curve with different combinations of weighting factors (figure 2.11). Although this approach can work for convex Pareto-fronts, it does not work for non-convex cases since not all points on the Pareto-front can be determined. It is apparent from figure 5.38 that many

points in the non-convex case will never be reached with any combination of the weights and the resulting optima are unevenly distributed.

Figure 2.11:

Sequential variation of weighting factors can be used to find trade-off solutions.



Utility

Another scalarization approach is the utility functions approach which is based on the general formulations of *utility theory*. Utility functions may be developed using engineering judgment or a more quantitative approach. The range of the utility function covers a range of acceptable alternatives. Most scalarization approaches can be represented via the utility function approach (de Weck, 2004).

A mathematical construction of a utility function allows non-linear combinations of objectives via intermediate utility functions, which are then combined into an overall utility function that will serve as a single objective. The method assigns costs to each objective, converting everything to minimum cost (Papalambros and Wilde, 2000). The method normalizes the utility functions. This provides for a mediating capability by translating diverse criteria into a common scale (Kockler et al., 1990).

Utility functions have been classified by various researchers into their most prevalent shapes (Cook 1997, Messac 2000). For example a larger-is-better or smaller-is-better relationship is represented by a monotonically increasing or decreasing relationship between the objective *Ji* and its corresponding utility *Ui*, whereas a nominal-is-better or in range-is-better type of utility can be represented by a convex or concave functions (figure 2.12).

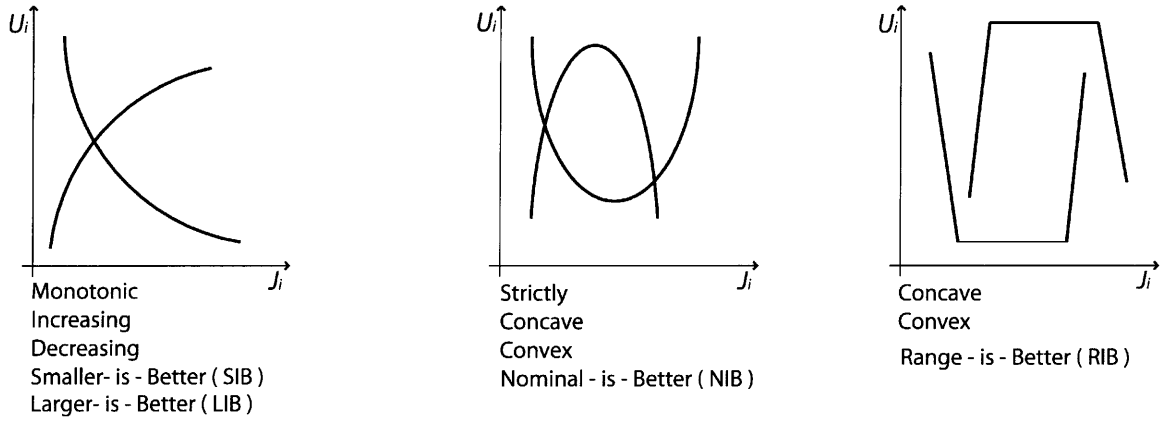


Figure 2.12:

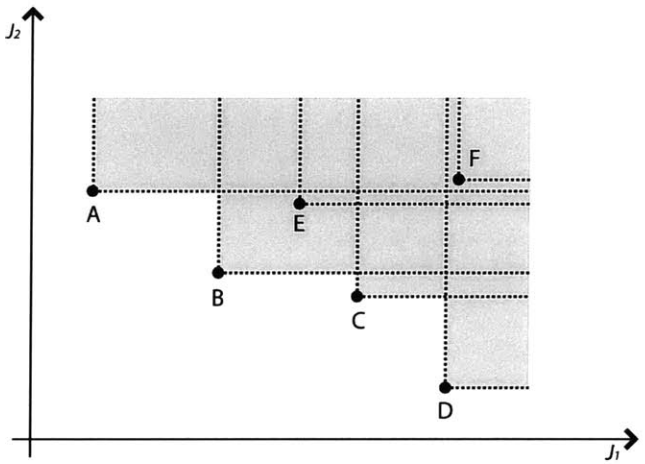
Different utility functions classifications (Cook 1997, Messac 2000).

Search Before Decision Making

A solution may be better, worse or indifferent to other solutions, neither dominating nor dominated with respect to the objective values (Abraham et al., 2005). In a multi-objective optimization problem there exists a set of solutions which are superior to the rest of the solutions in the search space when all objectives are considered but inferior to other solutions in the space in at least one objective. These are optimal solutions that are not dominated by any other solution in the search space (figure 2.13). Such optimal solutions are called Pareto optimal, and the entire set of such optimal trade-offs solutions is called the Pareto optimal set, where the rest of the solutions are called dominated solutions (Abraham et al., 2005).

Figure 2.13:

Points A,B,C and D are optimal solutions that are not dominated by any other solution in the search space.



All elements in the Pareto optimal set define reasonable solutions and are subject to further decision factors in order to choose a design for a given problem (Gries, 2004). As evident, in a real world situation a decision-making (trade-off) process is required to obtain the optimal solution (Abraham et al., 2005).

Although there are several methods to approach a multiobjective optimization problem, most work is concentrated on the approximation of the Pareto set (Abraham et al., 2005).

Pareto methods attempt to find a set of efficient solutions, x^{*j}, such that the objective vectors corresponding to those solutions are non-dominated in the objective space.

To explain the Pareto criterion for dominance we will assume, without loss of generality, two feasible objective vectors J^1 and J^2. For all objectives, respectively, J^1 dominants J^2 if and only if:

$$J_i^1 \geq J_i^2 \quad \forall\, i$$

$$\text{And} \quad J_i^1 \geq J_i^2 \quad \text{for at least one } i$$

This means a dominant solution is at least better in one objective while being at least the same in all other objectives.

For strong (strict) dominance requires J^1 to be better in all objectives than J^2.

Based on the notion of dominance, the simplest approach address the multi object decision is a combination of design space exploration and dominance (Pareto) filtering.

The advantage of multi-objective optimization compared to single objective optimization is to provide different solutions to the design problem that the designer can choose from. To pick one solution over another might require problem knowledge and additional decision criteria which are not necessarily formulated in the design task. Therefore, it may be useful to have a wide range of non-dominated solutions from which one or more solutions can be chosen.

Two goals can be pursued simultaneously in multi-objective optimization (Deb, 2001). The first goal is to find a diverse set of solutions. However, this set won't be comprehensive due to the *n*-dimensionality of the design vector *x*.

The Second goal is to find a set of solutions as close as possible to the Pareto-optimal front. Given that the points only satisfy non-dominance, the solutions obtained are only approximations of the Pareto Front.

An optimum to the problem is found if they satisfy the multi-objective version of the Karush-Kuhn-Tucker (KKT) optimality conditions (de Weck, 2004):

If x^* is non-inferior (=Pareto optimal) it satisfies the following KKT conditions:

a) x^* is feasible, i.e. $x^* \in S$ and $S = \emptyset$
b) All objective functions J_i and constraints g_j are differentiable
c) At x^* the constraints are satisfied $g_i(x^*) \leq 0 \; \forall \, j =$

$1,2,\ldots,m$ and $\lambda_j g_j(x^*) = 0$ whereby $\lambda_j \geq 0 \; \forall \; j = 1,\ldots,m$

d) There exist $\mu_i \geq 0 \; \forall \; i = 1,\ldots,n$ with strict inequality holding for at least one i such that the condition $\sum_{i=1}^{n} \mu_i \nabla J_i(x^*) + \sum_{j=1}^{m} \lambda_j \nabla_{g_j}(x^*) = 0$ is true.

The condition described in (d) expresses the fact that the gradients of the objectives and gradients of the constraints are in equilibrium with each other at a Pareto-optimal point. Note, that among multipliers, the preferences μi are the corollary to the weights (λi), while the λj's are the Lagrange multipliers.

2.2.4 Optimization Models

This type of mathematical model enables moving from one configuration to the other in the ongoing search for better solutions, but more importantly it is established with the aim of control and guidance.

Optimization techniques are often used to determine potential design configurations by optimizing them according to the functional objectives and requirements developed in evaluation models. The solution to the problem is generally developed through solving the mathematical model which consists of an objective function that is to be optimized, and a group of constraints that act as resource limitations (Bahrami and Dagli, 1994). Optimization usually offers crucial solutions in situations where design problems can be formulated according to the objective and functional requirements. In this optimization process, simulation can be used in computing variables of the design vector. If not appropriate, variables of the design can be altered and the process is then repeated.

Optimization was first coined with the development of the gradient steepest descent algorithm by Guass. It served as the first building block of the science of optimization. Later, in the 1940s, George Dantzig invented the term linear programming

which gave way to the development of the remaining well-known optimization schemes (Elster, 1993). Throughout the 1970s and 1980s, the field of Artificial Intelligence (AI) introduced the heuristic approach to solving optimization problems. Today, optimization plays an important role in most of the major fields, which include engineering and design, operations research and economics.

Conventional design procedures are for finding a suitable design which satisfies the functional objective(s) and requirements of the problem. In general, there will be more than one acceptable design or design alternative. The purpose of optimization is to evaluate and choose the fittest of the available acceptable designs based on the functional objective(s) and the design requirements and restrictions.

Optimization can be explicated as improving or fine-tuning a design or system in terms of one or more performance criteria (Papalambros, 2000). It formalizes what humans have always done intelligently. Optimization can be used in refining any design or system that includes some form of an analysis component, and is therefore subjected to the same limitations of the design. Generally, an optimization problem consists the following (Papalambros and Wilde, 2000):

- A set of variables that describe the design alternatives.
- An objective function(s), expressed by the design variables, to minimize or maximize.
- A set of constraints, expressed in terms of the design variables, to be satisfied by any suitable design.
- A set of values for the design variables, which satisfies all the constraints.

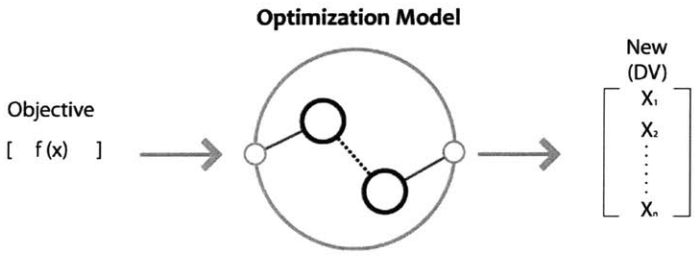
Certain design features are determined in the synthesis model, and the behavior corresponding to each design is determined in the analysis model. The evaluation model attempts to handle the multi-objective criteria of the design problem. Optimizing

models are then used to determine optimal designs.

The input to an optimizing model is an objective function. This could be the output of an evaluation model. The output of an optimization model is a new design vector that in turn is the input to the synthesis model (figure 2.14).

Figure 2.14:

Expected input and output of the optimization model.



The rising demand for industry to lower production costs has encouraged professionals to seek precise and accurate means for decision-making, leading them to utilize new optimization schemes. Optimization methods today have reached a high degree of sophistication, contributing to their use in a wide range of industries. With the rapid advancement of computer technology, the size and the complexity of the problems being solved using optimization techniques are also increasing.

In this section, optimization will be discussed in general. I will start by explaining how optimization problems are mathematically formulated and classified. Next, the main optimization algorithms will be discussed.

Mathematically, optimization is the minimization or maximization of a function subject to constraints on its variables (Nocedal and Wright, 2000). The objective function is sometimes called a "cost" function, since minimum cost is often taken to characterize the "best" design. In general, the criterion (objective function) for selection of the optimal design is a function of the design variables of the model.

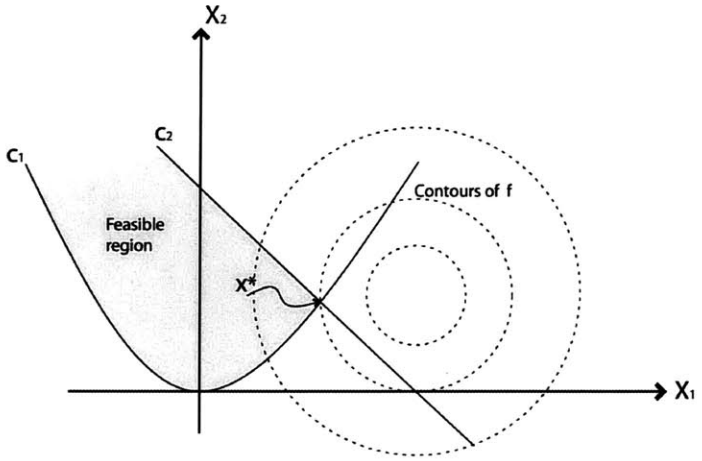
The following notations are used to represent a model (Papalambros and Wilde, 2000):

- *x is* the vector of *variables*, also called *unknowns* or *parameters.*
- *f* is the *objective function*, a (scalar) function of *x* that is maximized or minimized.

c_j are *constraint* functions, which are scalar functions of *x* that define certain equations and inequalities that the unknown vector *x* must satisfy.

Figure 2.15:

An optimization problem has an objective function and can have several constraints to insure feasibility.



Using this notation, the optimization problem can be written as follows:

$$\min_{x \in R^n} f(x) \text{ subject to}$$

$$c_j(x) \geq 0, \qquad i \in I$$
$$c_j(x) = 0, \qquad i \in E$$

Here *E* and *I* are sets of indices for equality and inequality constraints, respectively.

The variables are expected to be interrelated by physical laws, like the conservation of mass or energy, Kirchhoff's voltage and current laws, or other system equalities that must be

satisfied (Antoniou et al., 2007). Similarly a collection of constraints may be imposed on the variables to ensure physical reliability, compatibility, or even to simplify the modeling of the problem (Nocedal and Wright, 2000) (figure 2.15).

The classification of an optimization problem depends on more than one factor. It can be the objective function, constraints, design variables etc.

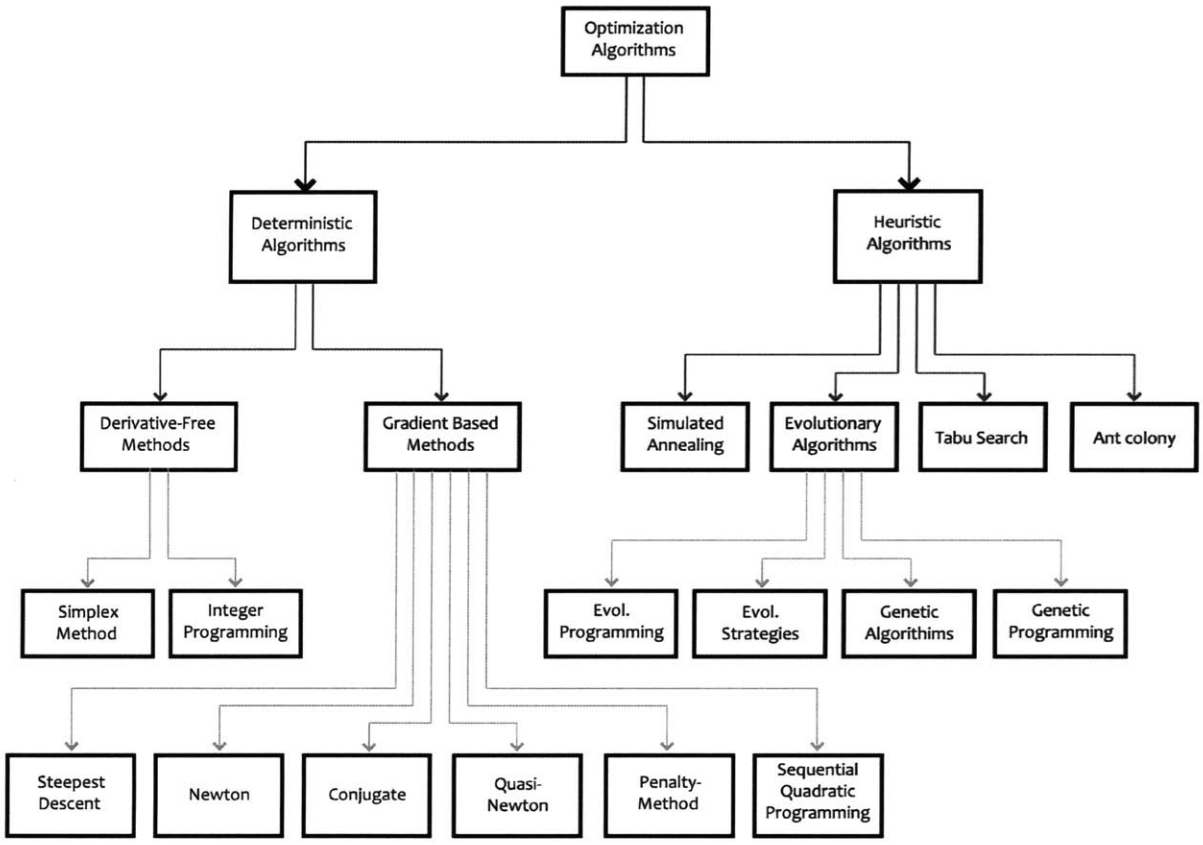


Figure 2.16:

A simple taxonomy of optimization algorithms discussed in the thesis.

Optimization Algorithms can be classified into either Deterministic or Stochastic (Heuristic) methods. Deterministic methods can be classified into Derivative-Free methods and Gradient Based methods. Stochastic (Heuristic) methods include several algorithms such as Evolutionary Algorithms, Simulated Annealing and Tabu Search (figure 2.16).

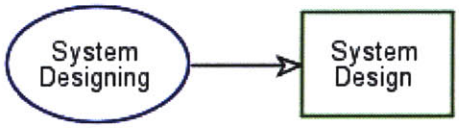
3. System Design and Designing

Objects and processes are the fundamental building blocks of systems. These building blocks can be combined in a variety of ways to model many complex systems, which are still objects and processes (Dori, 2002).

In addition, all systems in our surrounding environment are the result of designing. Design can imply many meanings. The word "Design" can be both a noun and a verb. Design as a noun can refer to the designed *object* while design as a verb can refer to the actual processes that generate the system's design.

Figure 3.1:

System Designing yields System Design



3.1. Design as an Object

As mentioned above, design as a noun can refer to an object or a system that can be defined by its geometric configuration, materials used, or the functions it performs. According to Bahrami and Dagli (1994), design can even be a developed plan or scheme, whether it is just embedded in the mind of the designer or externalized as a drawing or model.

Each one of us is a system object. We live within system objects and are surrounded by them. These system objects exist in nature as well as in virtually any conceivable area of human activity.

A system can be broadly defined as any product that accepts inputs and delivers outputs (Chapman et.al, 1992). Parts of these systems are sometimes systems in themselves.

A system in general has a group of basic characteristics. Any system should satisfy certain *functions* and consist of *components* that are the physical or abstract parts, elements or variables within the system. It also consists of *attributes* which define the properties of the system and its objects, and *internal structure* among its objects. In addition, any system exists in a *context*.

3.2. Design as Process

Design, as a verb, can refer to the actual processes involved in the decision-making activities that generate the system's design. These processes determine an object's form according to the required functions. Simon (1973) viewed design as a problem-solving process, a natural human activity that involves searching through a state space. The states in this space represent the design solution.

Coyne et al. (1990) define design as a purposeful activity that involves conscious efforts to reach a state of affairs in which specific characteristics are apparent. In this regard, design is initiated by recognizing the basic problem requirements. Being discontent with the existing state of affairs, the designer then becomes conscious that some sort of action should occur to correct the problem.

Louis Kahn, the famous architect, described design as a process where the inspirational forms of thinking and feeling generate form realization (Bahrami and Dagli, 1994). To Kahn, thinking was considered a tool by which he would articulate feeling, his ideal mode of functioning, into expressive shape. He believed that the design process was understood intuitively by the creative mind as a single unified and consistent whole,

as he used to synthesize elements from many sources into this whole rather than focusing on details of specific problems. He would pay more attention and dive into the core of the matter rather than going into finer-grain problems that were not really required at that point (Tyng, 1984).

Design as a process can be described as a systematic approach where the design process, as part of generating a system, is partitioned into general working levels. This allows for a transparent design approach that is both rational and independent of any particular field or industry. In this general approach, the problem is first analyzed, understood and decomposed into sub-problems. Sub-solutions are then generated and integrated to produce an overall solution (Cross, 1989).

There are several methods, intellectual frameworks, and tools that help support this process, including traditional engineering design (Pahl and Beitz, 1991), axiomatic design (Suh, 1990), and product design and development (Ulrich and Eppinger, 2000).

According to Papalambros and Wilde (2000) design is a complex human process that cannot be easily or completely described or understood. Therefore in the following chapters, I will use models to help define and understand the design process. Models are different from theories. According to Dixon (1987), a model does not establish a theory, but rather the theory is established when model behavior can be robustly explained through testing. They are however content with the provided explanations and predictions of phenomena, and can explain and sometimes replicate specific aspects of design behavior (Coyne et al., 1990).

In building useful models, the logical and mathematical relationships between components are required. It is easier for a designer to describe how a specific system is designed than

to translate his/her method into a logical or mathematical model unless a certain framework is developed for that purpose.

3.3. Towards an Evolutionary Design Model (EDM)

The design processes is an evolutionary process which occurs between the time a problem is assigned to the designer and the time the design is passed on to the manufacturer (Dasgupta, 1989). During this period the system design evolves and changes its form.

As discussed earlier, design can be seen as both an object and a process. The relationship between the designed *system* and the design *process* is represented within the *Evolutionary Design Model* (EDM).

EDM can be understood in the light of biological evolutions. The System's complex design, behaviors, structures, and requirements, their interfaces, and the progressive stages they undergo are all aspects found in biological evolutionary processes.

The EDM consists of a series of design evolution stages and processes. Each design stage in this evolutionary process is referred to as a *design state* and each set of processes is referred to as a *design evolving process*. The EDM serves as a meta-process. The system engineers create the design states and evolving processes on which it operates (Figure 3.2).

A design state describes a design at a particular point in time and maps the system object to the system's requirements and identifies its relation to the context in which the system will operate. A design state is a representation of the current level of understanding of design specification. *Requirements*, *context* and *system descriptions* must co-evolve as the design moves from a low degree of definition to a high degree of

specificity.

A design evolving process involves many sub-processes which include *formulation, decomposition, modeling,* and *integration*. These processes are not necessarily carried out in a sequential manner, but a continuous back and forth could exist between the different processes as the design progresses and evolves.

The formulation process identifies and boundaries of the problem to be solved and prioritizes the objectives to be addressed and formulates a general concept that guides the design solution.

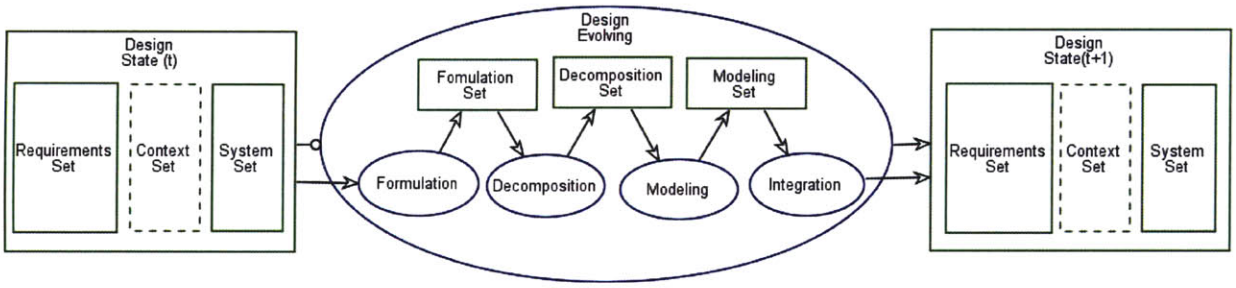


Figure 3.2:

The EDM consists of a Design State Object and a Design Evolving Process

In the decomposition process the system design concept is broken down using a functional-physical decomposition. These sub-problems facilitate, in addition to the problem solution, a better understanding of the design problem domain.

Modeling processes are constructed into hierarchical levels and design cycles in order to manage design complexities, where each lower level becomes more detailed and refined as the design progresses. The functional-physical decomposition informs the activity modeling process that include synthesis, analysis, evaluation and optimization

The Integration process then combines the design cycles and levels that converge to a new design state that should provide

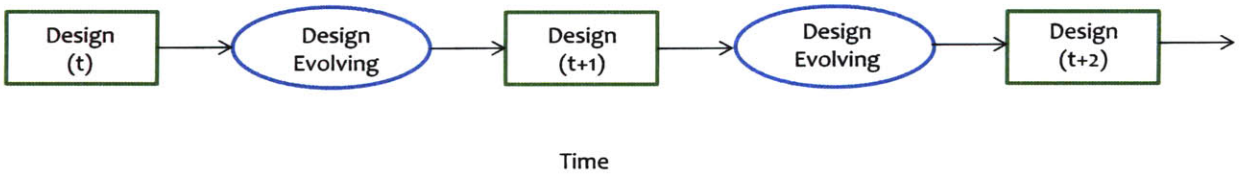
a more refined understanding of the systems system description, its requirements, and context.

The type of model used within the EDM evolving processes is highly dependent on and driven by design needs of each process. At the early EDM processes of Formulation and Decomposition the focus is on logical modeling Methods. Within the following stages of Modeling and Integration the focus is more on mathematical modeling approaches. As described earlier in chapter three, logical models represent logical structures and relations that do not necessarily have numerical values, whereas mathematical models can produce solutions and measure behaviors and performances numerically.

Furthermore, the EDM evolving processes can be implemented within a computational design system. The process of building computational models of design involves concepts from many disciplines such as artificial intelligence and problem solving, space search techniques, expert systems, neural networks, logic and fuzzy logic, object-oriented methodology, and language theory (Bahrami and Dagli, 1994). By implementing some of these concepts, a computational design system can be better defined, studied and understood.

Figure 3.3:

The design matures through states by going through several design evolving processes in the EDM design lifecycle



Within the EDM framework, the design matures through states *t*, *t+1*, *t+2* etc... by going through several design evolving processes in its design lifecycle (Figure 3.3). The resulting design model is described as evolutionary model that moves a system's design from simple abstract states to more complex

and detailed states throughout its evolution.

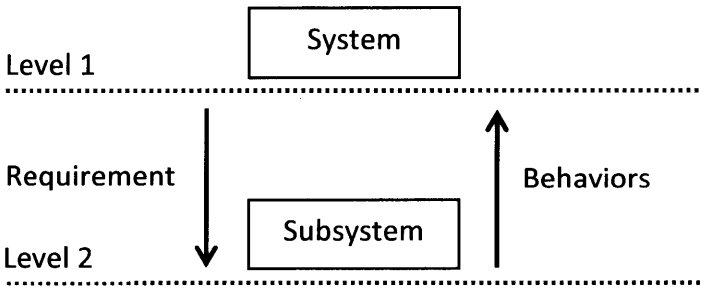
The proposed EDM can be described as encompassing an adaptive architecture which is affected by several factors. These include the number of design levels or cycles needed at a particular design level. Additional factors also include the required degree of fidelity of a specific design model.

In this model, the success of each step is demonstrated, and the basis for the next one validated, before decisions are made to proceed to the next phase. The main principle of EDM is that it continually builds on what already exists.

This process of evolution clearly involves the decisions of the system engineers on what requirements will be included and what processes will change over time. Effective planning is required, however, where there is an evolution from one state of maturity to the other. Hence, each evolving process should take uncertainty of later states into consideration.

Figure 3.4:

Design decisions at a certain level lead to requirements flow down and behaviors propagating upwards



When system designers make decisions at one design level, they establish requirements for the next level which leads to requirements flow down. In these decisions, designers constrain the design variables for subsequent decisions and design levels. Each decision also affects behaviors that propagate back up the hierarchy (figure 3.4).

4. Design State Object

Systems engineers usually use the concept of the state of a system for analysis and modeling of the behavior of a complex system that changes frequently over time. State can be considered a snapshot of a particular system at an instant of time (Sage and Armstrong, 2000). The main interest of the state theory approach is the description of the state of the system and the detection of changes in the system state according to new inputs.

If we consider the EDM as a system, then a design state will include three main sets that capture the designed system object and its relation to context and requirements. These sets are the Requirements set *R*, the Context set *C* and the designed System set *S*. The nature of these sets and the number of entries in each are closely coupled with the purpose being modeled and the complexity of the system being formulated. Using the three design sets a descriptive measure of the design state D_s can be provided at any given instant in time.

$$D_s = \{R, C, S\}$$

4.1. Requirements Set

Requirements are the cornerstone of the systems design and engineering process. Buede (2009) assumes four categories of requirements: input/output, technology and system-wide, tradeoff, and test requirements.

1- Input/ Output requirements:
The first category is input and output requirements. These

include inputs, outputs, interfaces, and functional and performance requirements that regulate the reception and delivery of input and outputs (Buede 2009).

Functional requirements are a set of all allowed inputs and possible outputs over time and the functional relationships linking them. Functional requirements do not convey any requirements in regards to the technology being used, or the process followed in the design (Chapman et al., 1992).

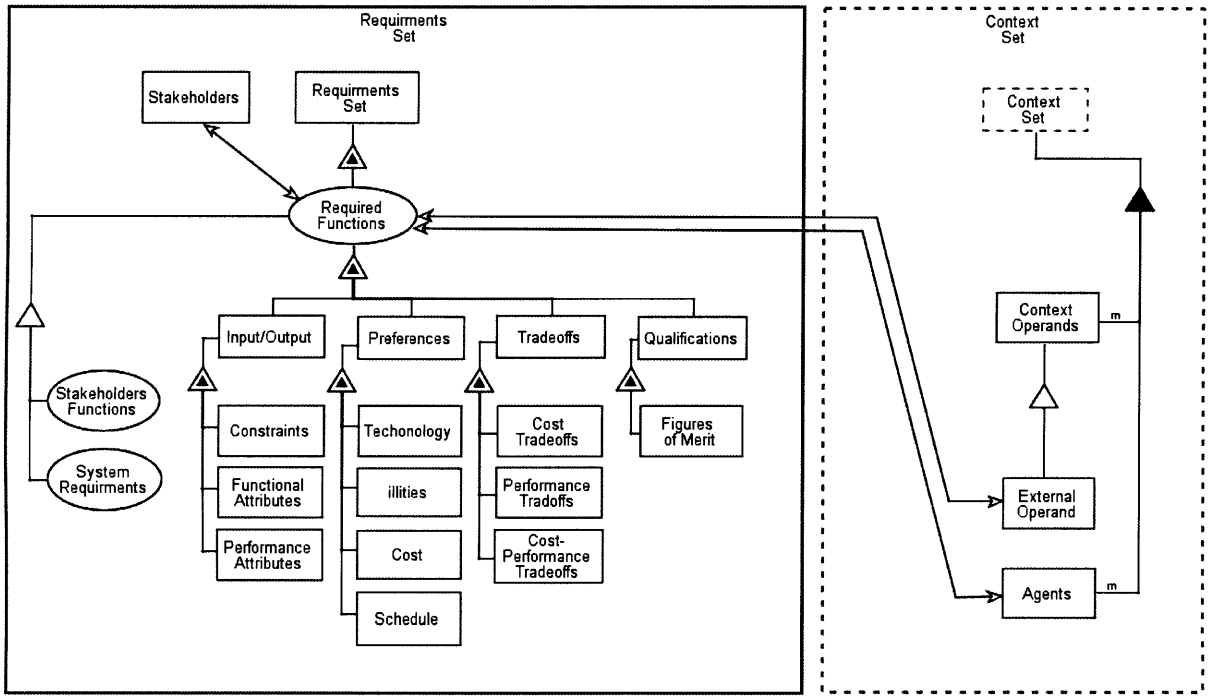


Figure 4.1:

Four categories of requirements: input/output, technology and system-wide, tradeoff, and test requirements.

Performance requirements describe how well the input/output and the functional relationships should meet the imposed requirements. Performance may be expressed in measurable terms called "figures of merit". Figures of merit should be measurable for any design in order to help make system design decisions (Chapman et al., 1992).

2- Technology and System-wide requirements:
Technology and system-wide requirements consist of

requirements addressing the technology to be integrated into the system, the "-itilities" of the system, cost, and schedule (Buede 2009). Technology requirements typically impose limitations, as defined by stakeholders, on technologies used in building the system. It dictates what processes should be used or eliminated, budgets, and schedule constraints (Chapman et al., 1992).

3- Trade-off requirements:
The third category of requirements describes the extents of allowed trade-offs including performance tradeoffs, cost tradeoffs, and cost-performance tradeoffs. Trade-off requirements specify the nature of trade-offs among input/output, system's technologies, and systems requirements. Trade-off requirements represent stakeholders' priorities as the base of selection within the design (Chapman et al., 1992).

4- System qualification requirements:
Systems test requirements specify methods for observing and testing the developed system in its final stage. Test requirements include specifications for estimating figures of merits to help approximate, analyze, and collected data when testing prototypes, systems models, and final system designs. (Chapman et al., 1992).

The above four categories of requirements are relevant to any D_s of the system's design life cycle. From a concurrent engineering perspective, each requirement category is to be used to address the relevant system in each life cycle D_s.

These requirement categories can be further mapped to both the initial stakeholders requirements and the evolving system requirements. There is an important distinction between the stakeholders' requirements and the system requirements.

Stakeholders' requirements R_{Stk} are those requirements that

the stakeholders provide through operational statements that define their needs.

System's requirements R_{sys}, on the other hand, emerge as the design evolves. Moving between successive D_s within the EDM progression establishes new system requirements for the following D_s. This leads to a requirements flow down. The design team cannot have a full description of the system's true requirements, its operational environment, emergent behaviors and projected future design decisions (Aughenbaugh and Paredis, 2004).

Progression of the design process, and decision making call for structuring requirements in hierarchies where each set of requirements is at the same level of granularity; one that is consistent with the current system-level.

4.2 Context Set

The Context set C is the set of entities that interact with the systems through its external interfaces. The context entities can impact the system as a form of input and be impacted by its outputs.

The context set could include operands in the context environment, external systems, or the system users. Context is responsible for some of the system's requirements and receives systems' output.

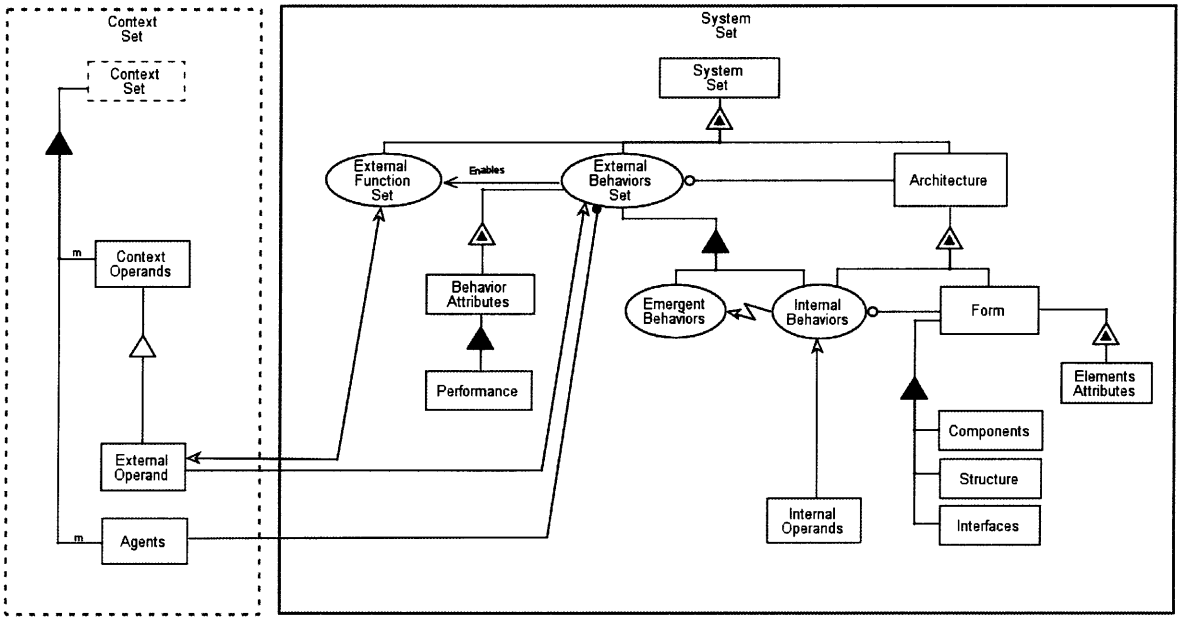
4.3 System Design Set

The System Design Set S includes related components S_c *and a structure* S_s, which allows for the interaction of the components through interfaces S_i. Together S_c, S_s and S_i comprise the system's *form* S_f. The system form executes certain system *Behaviors* B_s. The system's Behaviors include both *anticipated behaviors* B_a and *emergent behaviors* B_e. The

system behaviors should enable the *system functions* F_s. The combination of S_f and B_a defines the system's *architecture* S_a. Each of these sets should include elements at the level of granularity available at that level of the design (Figure 4.2).

Figure 4.2:

System Set Diagram



4.3.1 Functions

Function is essential to the definition of a system. The function of a system satisfies the purpose, the need or the objective for which the system is employed.

According to Eggert (2004), the function of a product is what it is expected to perform. Crawley (2003) defines function as a system attribute, conceived by the architect that denotes the activities, operations and transformations that cause, create or contribute to performance and meet the required goals.

Function can be considered as an attribute of system that

describes the rationale behind its existence, the intent for which is was built and the purpose for which it exists, the goal it serves. (Dori, 2002).

These definitions of function apply to both natural and artificial systems. The definitions emphasize what the system does and why it does it, and are not concerned with how it does it. Natural systems exhibit natural phenomena. Artificial systems are utilized for a purpose, aim or goal. These intents, goals and purposes are the basis from which the function of an artificial system is derived. Therefore, function describes what the system does and can do.

Crawley (2000) also defines function as "the actions for which a thing exists or is employed." In this definition Crawley defines the basis of function as being the goal. This emphasis on goals distinguishes function from behavior. Behavior describes how a system operates. Function on the other hand is concerned with what the system does and why it does it.

Form, as will be discussed later, is intimately related to function. The quotation by the famous architect Louis Sullivan, "Form ever follows function", supports this idea that the form of an object is highly dependent upon the function it performs. Similarly function is associated with form and emerges as form is assembled, as well as when different sub-functions are aggregated together yielding what the whole system eventually "does" (Crawley, 2003).

It is important in the design of complex systems to understand that these systems have specific *primary functions*, in addition to other properties known as *ilities* (Crawley et al., 2004) which include adaptability, durability, maintainability, flexibility, etc. Primary functions denote the immediate value of a system, such as flying for airplanes, delivering products for companies, and so on. Ilities, on the other hand, have life-cycle value that describes properties of "performing things well".

4.3.2 Behaviors

The behavior of a system describes how an object operates to achieve the function. Function and behavior are not synonymous. They are two different terms. Normally, what we may perceive as the function of the system is actually its behavior. In fact it is the main process that the system carries out that facilitates for a system its function. As mentioned earlier the function is derived from the system's goal; it is what the system does and why it does it. On the other hand, behavior is how the system behaves to achieve the system's function (Dori, 2002)

The term "behavior", according to Eggert (2004) describes how a product actually performs. The aim of system design in general, and of architectural design in particular, is to achieve the desired behaviors that are outputs of functions plus ilities while predicting and limiting undesired behaviors.

Some system behaviors are considered to be deliberate and intentionally developed through methodical design activity. These are *anticipated behaviors* and can be desirable or undesirable.

The system behaviors can be organized by the degree of complexity of subsystem interactions, and how they affect the behavior of the system as they aggregate from the subsystems to higher levels. Anticipated behaviors can be seen as dependent on the system components or on system structure.

Components dependent behaviors are found in systems that exhibit simple relationships. In such systems, subsystem behaviors easily aggregate from the level of the components to the level of the system. The system's behavior only depends on the breakdown of the system into subsystems, and how these subsystems are configured and behave.

Structure dependent behaviors are best illustrated through cost since they are more complex than component dependent behaviors. The total cost of a system is not the sum of its parts. Instead it also includes costs that depend on the structure of the system. When a system is assembled there is a cost which is not component dependent but structure dependent. There it is said that the system's structure affects the cost of combining two or more subsystems. (Aughenbaugh and Paredis, 2004).

Complex systems have behaviors that are usually not attributed to their individual sub-components. In their definition of a system Chen and Stroup (1993) note that a system is "an ensemble of interacting parts, the sum of which exhibits behavior not localized in its constituent parts." This definition proclaims that the behavior of the whole system is different than that of the parts. It alludes to the principle of synergy, where the whole is more than the sum of the parts. This proclamation is indeed true when studying systems with *emergent behavior* (Dori, 2002). As such the definition implies that the behavior cannot always be attributed to any one part of the system.

These unanticipated emergent behaviors are very similar to what Ulrich and Eppinger (2000) identified as "incidental interactions". They exist when the system or its interactions with the surrounding context are not fully comprehended. They can exist due to other unpredictable factors, such as future system changes or the difficulty of modeling every single system state. Emergent behaviors can be desirable or undesirable when thought of in retrospect.

Once operational attributes that capture the illities of the system are accounted for within a model of the system, the complexity of the system will increase. The interactions between the components and the structures of the system are made more and increasingly complex by introducing operational attributes of the system. The interactions between

subsystems and components increase. Interactions resulting from operational attributes are just as important as the internal workings of individual systems.

Anticipated and emergent behaviors need to be estimated during system design. As the complexity of systems increase, advanced tools such as modeling and simulation becomes vital to be able to estimate the system's behavior with a closer degree of certainty. To realize a good behavioral estimate of a system, most of the behaviors within the subsystems must be identified and estimated.

By controlling and constraining the behaviors of subsystems, designers can produce the desired system level behavior. However, in order to make the decision about which subsystem behaviors to constrain, designers must explore the full range of emergent behavior of the system.
It is also important to discuss the relation of performance to behavior and function. Performance is an attribute of a system that measures the effectiveness of the system's function.

Dori (2002) gives an interesting example of a system for adding numbers. The performance of the number addition system can be measured by speed, accuracy and error rate. Three different architectures of an addition system are an abacus, a hand-held calculator and a laptop computer. As an addition system, the performance of the hand-held calculator is the highest among the three apparatuses.

4.3.2 Operands

Operands represent the media generated, sent and received within the system, between one part of the system and the other, and ultimately assist in the transformational process of the system.

There are three fundamental operands that compose the

media on which systems operate and function. Operands can be physical entities with material and energy. Material refers to the substance of physical objects, while energy boosts the operation of the active system components.

Operands may also be entities of information, which refers to knowledge content and communication that are somehow transformed into entities of material or energy and transmitted between other physical entities.

The physical embodiment of individual functional elements is thus usually configured through the construction of material, the control of external information, and the power of a source of energy, regardless of the primary function and classification.

Information can be further subdivided into two classes. The first class involves *signal elements* that sense and communicate information, such as radio signals. The second class involves *data elements* that interpret, analyze, organize and manipulate information, such as computer programs (Kossiakoff and Sweet, 2002).

4.3.3 Form

The determination of form to satisfy and execute a required function represents the essence of design. Eggert (2004) defines form as what the product looks like, what materials it is made of, and how it is made. He identifies the basic characteristics of the form of a product to be shape, size, configuration, material, and the manufacturing processes used to make the product.

Form, according to Crawley (2003), refers to the physical or informational embodiment that exists or has the potential to exist. Form represents the thing that is eventually implemented and operated in a solution specific domain. Implementation here can include manufacturing, building, writing, composing,

etc. Operation can refer to running, repairing, updating, etc.

Form can be represented as the sum of components (objects) structure, and interfaces. Components are segments of the whole of the form, and structure denotes the formal relationships among the objects, while interfaces represent the crossing points between components and structures.

4.3.3.1 Components

A component is a subset of the physical realization of a system. The component in a system is allocated a subset of the system's function. A component could be an integration of hardware and software, a specific piece of hardware, a specific segment of the system's software, a group of people, facilities, or a combination of all of these. There is a hierarchical structure for components in a system.

4.3.3.2 Structure

Structure describes the assembly of components within the system. It describes the long-term relationship between the components that is unaffected by the flow of time. In a system interactions would exist between at least two subsets. There must be an emphasis on the internal interaction among the system's components. In a possible mathematical model of the system proposed by Wand and Weber (1989), "if we view such a set of things as a graph, where everything is a node and every interaction is represented as a link, then a system is a connected graph." Thus any two things in a system must be related, directly or indirectly. If a system only contains two things, the connectivity between them is indeed important is if the assembly were to be called a system. If there is not even one structural link between the two things, then it is not possible to claim any relationship between the things.

Structure describes the relationships among system

components (Crawley, 2003). It can describe connections that take place both in form and in behavior while operating. Connections that are descriptions of form include concepts of spatial location, proximity, topology, or assembly process. Connections that are descriptions of behavior include flow of information, energy, and material. Products and systems are separated from other supporting systems and operands by a boundary.

4.3.3.3 Interfaces

An interface is a connection resource of a system which could be both internal and external. The system's interface is called an external interface if it hooks into another system's interface. An internal interface in which a component hooks into another component within the system. Interfaces operate functions and take inputs and produce outputs. Interfaces represent common sources of failure within system.

4.3.4 System Architecture

Every system has an architecture, which in essence strongly affects its behavior (Crawley et al., 2004). Architecture is significant in a variety of disciplines and in many technical fields. The typical connotation describes civil architecture of buildings, but the term also extends to include physical products, engineering systems, and infrastructures, in addition to informational artifacts such as software and computer networks.

"Architecture" in Webster's Online Dictionary is a "formation or construction resulting from or as if from a conscious act," or "a unifying or coherent form or structure". Crawley (2003) describes a generic architecture as "the conceptualization, description, and design of a system, its components, their interfaces and relationships with internal and external entities, as they evolve over time".

There are multiple definitions for "architecture" which are largely dependent on the context in hand. Different disciplines look at architecture from different perspectives. Such disciplines include product development, mechanical systems, engineering systems and others. From a product development viewpoint, for example, architecture is described by Ulrich and Eppinger (2000) to be an "arrangement of the functional elements into physical blocks". In the engineering systems field, system architecture is defined by the ESD Architecture Committee at MIT as "an abstract description of the entities of a system and the relationships between those entities" (Crawley et al., 2004). They also state that architecture, which embraces meanings such as an "arrangement of entities and relationships between them" or as "relationship between form and function", represents the physical embodiment that the designer finds in order to perform the required functions of the design problem.

The Open Group Architectural Framework (2001) defines architecture as "a set of elements depicted in an architectural model and a specification of how these elements are connected to meet the overall requirements of an information system." The definition is in the context of information systems. It emphasizes that the connections between elements need to be specified. This represents a structural aspect fundamental to the integrity of the architecture.

These definitions share many things in common. The basic common characteristics of architecture include the description of the system components, the structure of the interrelationships among them, and the functional character of these components and their interrelations. Every discipline, however, differs in terms of the specifics of what those parts are and how accurately they are connected together.

The system functions are achieved with a particular

combination of form and behavior. Behavior is intimately related to form. Behavior represents what happens to a form and represents the sequence of occurrences. The successful combination defines the system's architecture. A system's function can be facilitated by any number of different architectures (form-behavior combinations). Hence, Dori (2002) defines system architecture as the overall system's form-behavior combination, which enables it to attain its function while embodying the architect's concept.

Defining an architecture for a system serves many goals, such as abstraction, reducing the impact of continuous changes, and facilitating communication (Zachman, 1987). An architecture abstracts complex systems through describing simple models. This abstraction enables the definition and control of interfaces and the integration of system components. An architecture also enables reducing the impact of changes to fewer steps especially in redesign processes. It focuses on parts that require major change. An architecture offers multiple abstract views on the system and provides a means of communication during the design or re-design process, where useful discussion occurs to represent the perception of each communicating party of the problem in hand.

Perry and Wolf (1992) draw an analogy between system architecture and the architecture of buildings. They describe how architecture provides multiple views, abstractions, architectural styles, and how engineering principles and materials significantly affect the architecture of a building. A building architect's interaction with a client versus a contractor for example, the architect provides different views of the building in which there is a focus on some specific aspect. He provides elevations and floor plans in addition to scale models for the client in order to give him a good impression of the building. The contractor however is provided with the same floor plans in addition to structural views that provide detailed information about diverse design considerations.

Architectures can arise within a variety of mechanisms (Crawley et al., 2004). These include the deliberate design of a system from scratch, the evolution of a design from previous designs with strong legacy constraints, obeying regulations, standards, and protocols, the expansion of smaller systems with their own architectures, or the exploration of form and behavior requirements through dialogue between architects and users.

Mapping function and behavior to physical elements (Form) within hierarchical structures is significant in system design. In a top down approach, the system's function dictates and delimits the range of combinations of form and behavior, thus realizing the function of the system, and achieving the goal for which the system is designed. Alternatively, it is also of possible to reverse the hierarchy where a function can be the result of a particular form.

Ulrich and Eppinger (2000) define two categories of functional and behavioral mapping related to product architecture, which refers to the scheme by which functions and behaviors are mapped to physical elements and the internal interactions between those elements are defined. These categories are modular architectures and integral architectures.

The basic characteristic of modular architectures is the relatively strong and direct one-to-one mapping of functions to physical elements. Since the role that interfaces play for each function among the different physical elements is well defined, modular products become more appealing. Moreover, individual physical components can be designed relatively independently by functionally decoupling them. Downstream integration throughout the design process thus becomes less complex.

Integral architecture, however, involves a complex mapping of functions to physical elements. There is no direct mapping and the interfaces of physical elements acquire complex relations

to functions (Ulrich,1995). The consequent effects of element interactions on functions are hard to recognize, or *incidental* (Ulrich and Eppinger, 2000). Functions in *inherently integral* products are delivered in a coupled fashion, meaning that modifications in a part, feature, or sub-element of a product affect the global system performance in many functions. The term "inherently integral" is used here to refer to products that have many functions shared by many of the same physical elements.

Some design theory literature considers modular architecture ideal and considers a design to be inferior if designers could not achieve modular design. However, what occurs in reality implies that designs with integral characteristics can represent a higher degree of success and goal accomplishment by their designers (Ulrich and Seering, 1990; Whitney, 1996). In a real design problem, designers are faced with many goals to achieve. These goals often conflict with each other and cannot all be attained equally well. It is important to consider the relevance and use of both modular and integral architectures in system design. Integral architectures can be deployed in the case of simple system (Ulrich and Ellison, 1999) and even in complex inherently integral system which do not conform to ideal models or where modularity is not desirable. Even when considering engineering issues, integral architectures are still relevant. Modular architectures are needed, however, and become more relevant in situations where strategic issues are included, such as outsourcing and new architecture development.

Therefore, in short, the modular scheme can be viewed as one which exhibits strategic goals such as additions, adaptation, flexible processes, and diversity of production (Ulrich, 1995), due to direct mapping. The integral scheme, however, accommodates better overall performance at the expense of strategies (Ulrich and Seering, 1990 ; Whitney, 1996).

5. The Design Evolving Process

The design evolving process evolves a design state from a higher more abstract design level into a lower more detailed design level.

The design evolving process includes four main sub-processes: *Formulation, Decomposition, Modeling,* and *Integration.* Although these processes may appear to be sequential with steps following each other, the reality is that certain knowledge can be gained or some circumstances can change as the process moves forward, thus questioning decisions early on in the process and therefore forcing a return to a previous process.

5.1 Design Formulation

The design formulation process is the first phase in the design evolving processes. It receives input from a previous design state and comprises of functions that assist in better defining the design problem. These include: defining the system boundary, developing the system's objectives hierarchy and the development of a design concept (Figure 5.1).

A System's boundary defines where a system starts and ends in relation to other systems. It also defines the inputs from and outputs to the boundary's external environment.

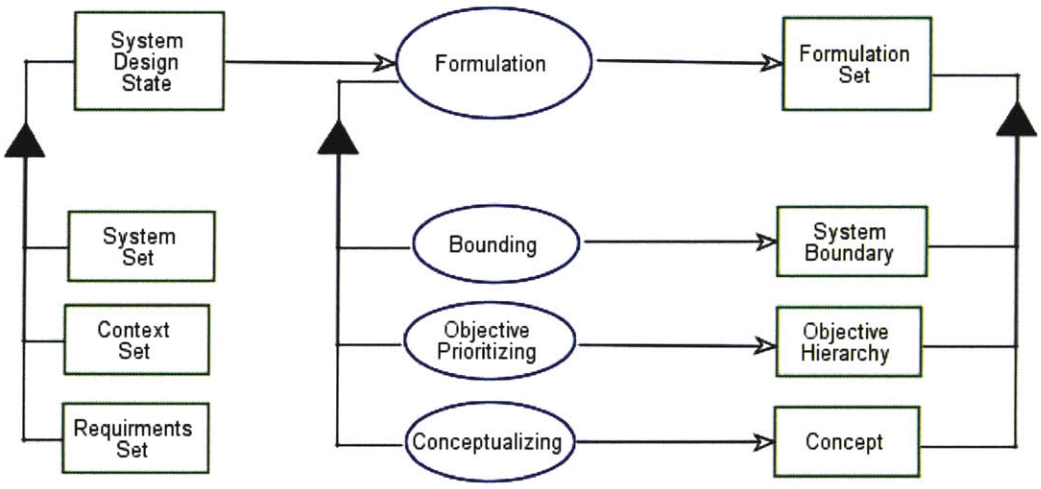
Developing the system's objectives hierarchy helps set priorities among objectives. These hierarchies are based on the stakeholders' needs and value system. The fulfillment of the defined objectives might not be possible due to conflicts.

Conflicting objectives define the very nature of design processes.

The design team should aim at developing a design concept that can guide the system design process, impose a context-specific vision, and ensure higher degrees of satisfaction based on the value systems defined by stakeholders'.

Figure 5.1:

Design Formulation Diagram



5.1.1 System Boundary

A *system boundary* around any subsystem is the entity that cuts across the links with the system environment and determines the input/output characterization.

Defining the system's boundaries is one of the most important steps in system design, since all the following design processes depend on it. This takes place by identifying all possible inputs a system consumes and all possible outputs it produces. Boundaries are also crucial in defining the system architecture as they identify the deliverables and responsibilities of the different design teams, and at the same time they define what exactly is fixed or constrained at the boundaries. Within the system's boundaries the nature of the system can change,

while the external environment is dealt with as having a fixed nature. Anything that crosses that boundary must be facilitated by an interface.

System's boundaries should not be perceived as fixed definitions, but rather as transient notions defined for each design phase of the system's life cycle.

In many cases, it may be more suitable and convenient to consider parts of the system as black box systems although there may be partial knowledge about them. This is due to the fact that in many design situations, the relevant and required information is more related to system performance as a whole in terms of the relationships between system inputs and outputs, rather than the complex interrelationships among internal system components and their own individual behavior (Meredith et al., 1985).

Once a clear and explicit relationship exists between a group of elements belonging to known input variables and another belonging to required solution output variables, a systems problem can be solved. The behavior of the black box system is depicted and analyzed in terms of the changing values over time of both groups. There is no need then for the knowledge of the internal structure of that system's components once the functional performance criteria for the input-output behavior have been established.

5.1.2 Objectives Hierarchy

The systems engineering design process is decision rich. As will be discussed later in this several times in this thesis, after identifying alternatives, decision making is required as part of the evaluation activities of the design process in which objectives and criteria are considered to evaluate the generated design alternatives. However, Keeney (1994) argues that this *alternative-focused thinking* of using decision analysis

to only evaluate alternatives is reactive rather than proactive.

Keeney proposes *value-focused thinking* that emphasizes structuring decisions in terms of fundamental objective that must be determined prior to solving the design problem. Keeney suggests that this value focused thinking leads to uncovering hidden objectives and better information collection. Implementing a value-focused thinking approach can help the earlier design activities of synthesis and analysis in generating better alternatives.

Therefore, a fundamental set of objectives should be identified early on in the process. The fundamental objectives of a system design are a collection of essential objectives defining design decision problems being dealt with. A hierarchy of fundamental objectives or value structure is achieved by breaking down and prioritizing objectives into sub-values (Buede 2009).

Objective hierarchies are therefore a hierarchical representation of key system characteristics as valued by stakeholders. These characteristics can include: performance aspects, cost, scheduling, etc. Value curves and weights can also be defined based on information gathered from stakeholders (Buede 2009).

This objectives hierarchy, curves and weights can lead to establishing value structures. These value structures can guide the system designers in design trade-offs studies when comparing among design alternatives. These alternatives compete with one another in having one characteristic better than others. By comparing possible trade-off schemes, system designers can identify the preferred design alternatives. It should also be noted that identifying the objective hierarchies is essential for each design phase in the system design life cycle.

5.1.3 Concept

A design concept is required for every phase of the design life cycle. These concepts build on and are constrained by previous design decisions. Design concepts provide substantial, yet incomplete, amount of information on the systems' interaction with other systems and the system context.

A design concept is a contextual vision for how a system operates. Concepts should aim at materializing stakeholders' expectations, and meeting their requirements (Buede 2009).

Requirements and expectations can be met by more than one concept. What should govern concept selection is concept's capacity in satisfying requirements. (Chapman et al., 1992).

A system design concept is a design pattern or a combination of several patterns that provide a solution for one or more of the design problems. Several concept alternatives can be generated using different pattern classes. These alternatives should undergo a comparison process after which the design team selects the best satisfactory concept (Chapman et al., 1992).

Best satisfactory concepts are recommended, and selected based on approximating, simulating, or measuring trade-off potentials.

The concept development processes involves the necessary analysis and planning to understand the needs or requirements for a certain design phase and the ultimate system foreseen to fulfill those requirements. Several approaches have been discussed to identify the detailed nature of this process (Sydenham, 2003; Kossiakoff and Sweet, 2002; Eggert, 2004).

Kossiakoff and Sweet (2002) embrace three main subdivisions: the analysis phase, the concept exploration phase, and the

concept definition phase. In the analysis phase, the basic needs and requirements for a new system are defined in a continuous search for a "practical approach" that can possibly satisfy those requirements. The concept exploration phase tends to formulate and validate specific performance requirements for a set of potential proposed concepts. This phase focuses on how these performance measures address the original requirements and sets a valid goal for a new product. This is done for all potential concepts before exerting major effort on individual development. The concept definition phase looks at key characteristics of the alternative concepts and selects the most beneficial in terms of performance, estimated cost, development and operational life. After defining the functional characteristics of the preferred concept, major resources are committed to carry this concept forward to subsequent phases of preliminary and detailed development.

The design concept will guide the following physical synthesis activities and system behavioral analysis in both the decomposition and modeling stages of the design evolving process.

5.2 Design Decomposition

When reaching a concept, designers start the process of functional and physical decomposition (figure 5.2). Functional-physical decomposition refers to hierarchically related subsystems presented schematically as a pyramid whose top is the higher-level system and base is the lower level subsystem. Subsystems may correspond to physical components, and in this case the decomposition is called *physical decomposition.* Subsystems can also correspond to functions and the engineering disciplines which contribute to the system design. In this case the decomposition is referred to as *functional decomposition.*

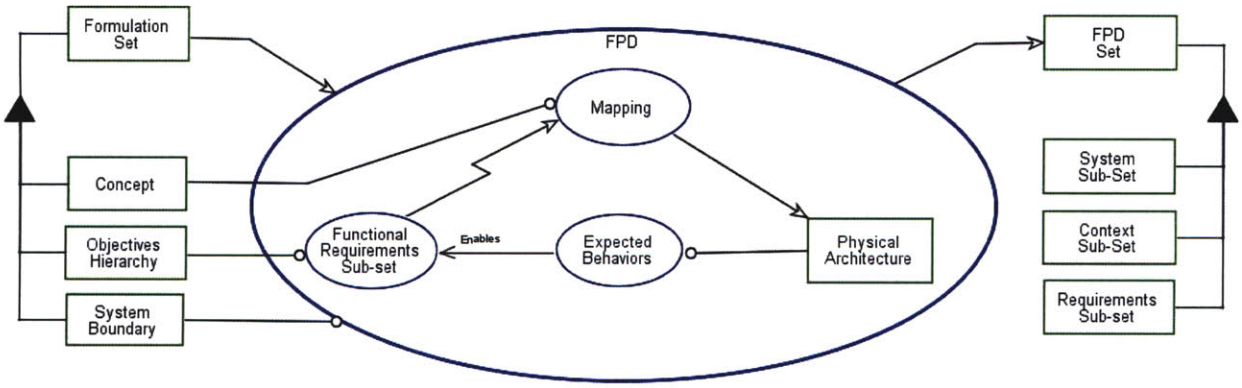
It is critical that the system design decomposition is developed

before hard modeling is carried out. Physical decomposition is essential to the synthesis activity later in the modeling phase. Functional decomposition, on the other hand, is important for the analysis activities and models.

The Axiomatic design (Suh, 1990) which attempts to map functional requirements to physical components, is a good design model that describes functional-physical decomposition. System object decomposition typically occurs in a top-level fashion, where first the conceptual design is partitioned into subfunctions. In parallel the system in its physical form is broken down into subsystems that can perform the subfunctions. Decomposition continues similarly until it reaches single parts. Throughout the process, design and testing of physical components are assigned to different disciplinary parties. This facilitates the synchronized development of different parts of the product by these parties.

Figure 5.2:

Design Decomposition Diagram



While top-level functions remain the same, they break out different when physical components are considered. Two different physical components can have the same function. Stakeholders' requirements such as cost performance, and reliability will determine which of the components is superior.

The proposed development decomposition uses different abstractions and levels of aggregation of complex interacting elements. Choices have to be made on the level of abstraction needed. High abstractions do not usually require domain knowledge, and are therefore used to summarize, generalize, and compare. Low abstractions require domain knowledge, and thus provide valid details where differences are explicable.

5.2.1 Functional Decomposition

Functional decomposition involves breaking down a problem according to its functions, which in turn can be assigned to a specific discipline that can handle the different physics of the system.

Functional decomposition starts at the top-level where Input/output and functional requirements are assessed based on stakeholders' requirements. When performing system's functional decomposition, designers need to decompose the top level functions into a set of functions following the same format of the overall system function found at the top level. This top down process aims at resolving functional requirements into smaller portions (simple tasks) that can collectively achieve the performance of the top-level function (Chapman et al., 1992).

System functions are defined at various stages and require a complete set of system requirements consistent with those defined at earlier stages. This process is known as requirements decomposition and allocation. The interrelations between sub functions require accurate and careful descriptions to ensure that the overall system function is satisfied (Chapman et al., 1992).

The act of decomposing in functional decomposition is oriented towards the different domains of knowledge involved in the design problem formulation rather than the physical

components. Functional decomposition usually divides the design system into well-defined categories, however, it may fail to account for disciplinary coupling.

5.2.2 Physical Decomposition

The process of physical decomposition and allocation is different from system functional decomposition. Physical decomposition breaks down the design in relation to the known physical parts (or components). Physical allocation of components should run concurrently with system functional decomposition. Decomposition of top-level system function is guided by physical decomposition. At each decomposition stage, designers need to describe interrelationships among systems components in great detail to perform the overall function (Chapman et al., 1992).

Physical allocation starts with demonstrating the validity of each system function by identifying an operational concept for its implementation. Then comes the assignment of physical elements to system functions.

The hierarchy of the decomposition is such that a system's physical elements are organized usually into physical building blocks called chunks. Chunks consist of a group of components that execute the functions required for the system.

Some physical elements become more defined, usually with design progress, while others are dictated by the system concept. The outcome of this kind of decomposition is affected by the approach selected for component decomposition, which in turn are influenced by the desired functions that should be performed.

Decomposition of systems relies on system designer's experience. It is a system in and of itself designed by engineers and architects attempting to solve design problems. It should

be noted that in system Functional-physical decomposition, it is not typical that designers persistently follow a top-down decomposition process to the level of single parts. They can also iterate between upper and lower system decomposition levels according to what they can potentially learn within the process about the implications of some of their architectural decisions.

It is important to note that designers must document the interrelations among physical elements and the interfaces between inputs and outputs. It is important to demonstrate that system requirements can be satisfied, all physical components are justified, and thus all system functions are performed (Chapman et al., 1992).

5.3 Design Modeling

As discussed earlier in the Systems modeling chapter, models are abstract descriptions of the real world that provide approximate representations of more complex functions of systems (Papalambros, 2000). Many design problems require using a group of complementary models, instead of one single model, which together aim at modeling and describing the whole design problem.

Complex systems design requires specialized knowledge in many disciplines (Pahng et al., 1997). No single designer can excel in all these disciplines, hence there is a need for different people who have the suitable principal competencies to model and solve different aspects of the design problem (Eppinger, et al. 1994).

In this chapter I will discuss two modeling modes: design process modeling, and design activity modeling.

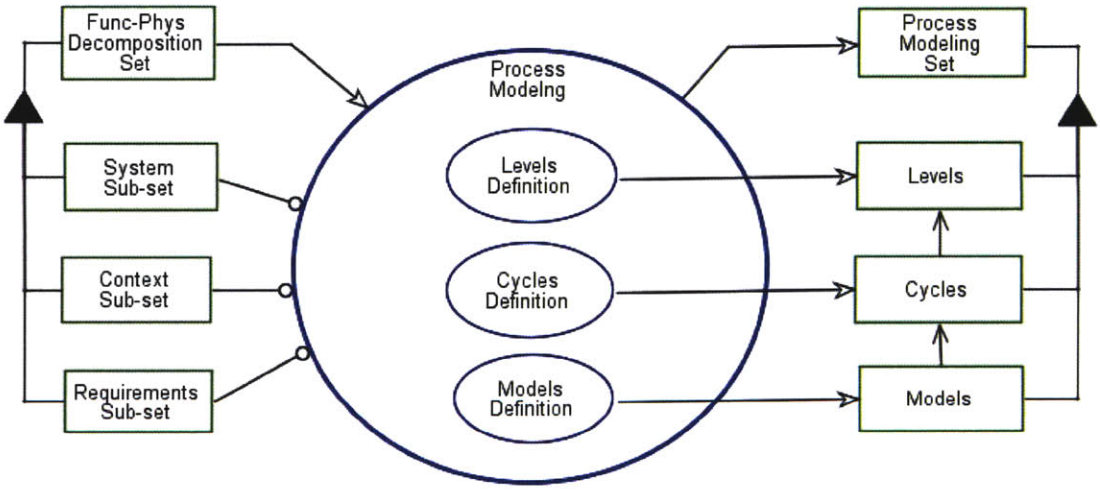
5.3.1 Process Modeling

Design process modeling is based on the fact that design processes comprise a number of smaller design activities. The design process can be modeled by tracing design information exchanged between different design activities.

Process modeling would typically take place before mathematical activity modeling and software programming to avoid major reprogramming later on. Process modeling basically promotes the interaction among the system architects, design specialists and other project members, as well as allowing the visualization of control flow and data.

Figure 5.3:

Process modeling
Building Blocks



Process modeling can proposed here uses several abstractions: design level, design cycle and design activity (figure 5.3). These abstraction categories offer a framework where the modeling of complex design problems can be achieved by aggregating sub-problems.

5.3.1.1 Design Levels

A system that comprises design activities with high complexity cannot be easily or efficiently designed as a monolithic entity, and so it has to be broken down into more manageable parts and hierarchies.

The modeling process can include both hierarchical and non-hierarchical structures. Within its hierarchical structure, it is possible to define discrete tree-like interaction patterns which offer well-guided navigation within the process. This hierarchical layout enables multilevel problem formulation. The design levels discussed here are of such a structure. These hierarchies can also be layered hierarchies where horizontal relations could be established within a single design level and between two or more design cycles. The non-hierarchical structures define relations between the different elements within a level. These elements include models and design cycles.

5.3.1.2 Design Cycles

Iteration is a basic concept in all design processes. Researchers have discussed different approaches for managing these iterations (Smith and Eppinger, 1997). I refer to iterations within the context of EDM as design cycles, were each cycle includes all four design activities discussed earlier, namely synthesis, analysis, evaluation, and optimization. Each design cycle resides within a design level and there could be several cycles within one level (Figure 5.4).

Through a bottom-up approach the design activity models are connected design cycle. The EDM as a whole can be seen as a set of interrelated models that collectively can produce system design solutions.

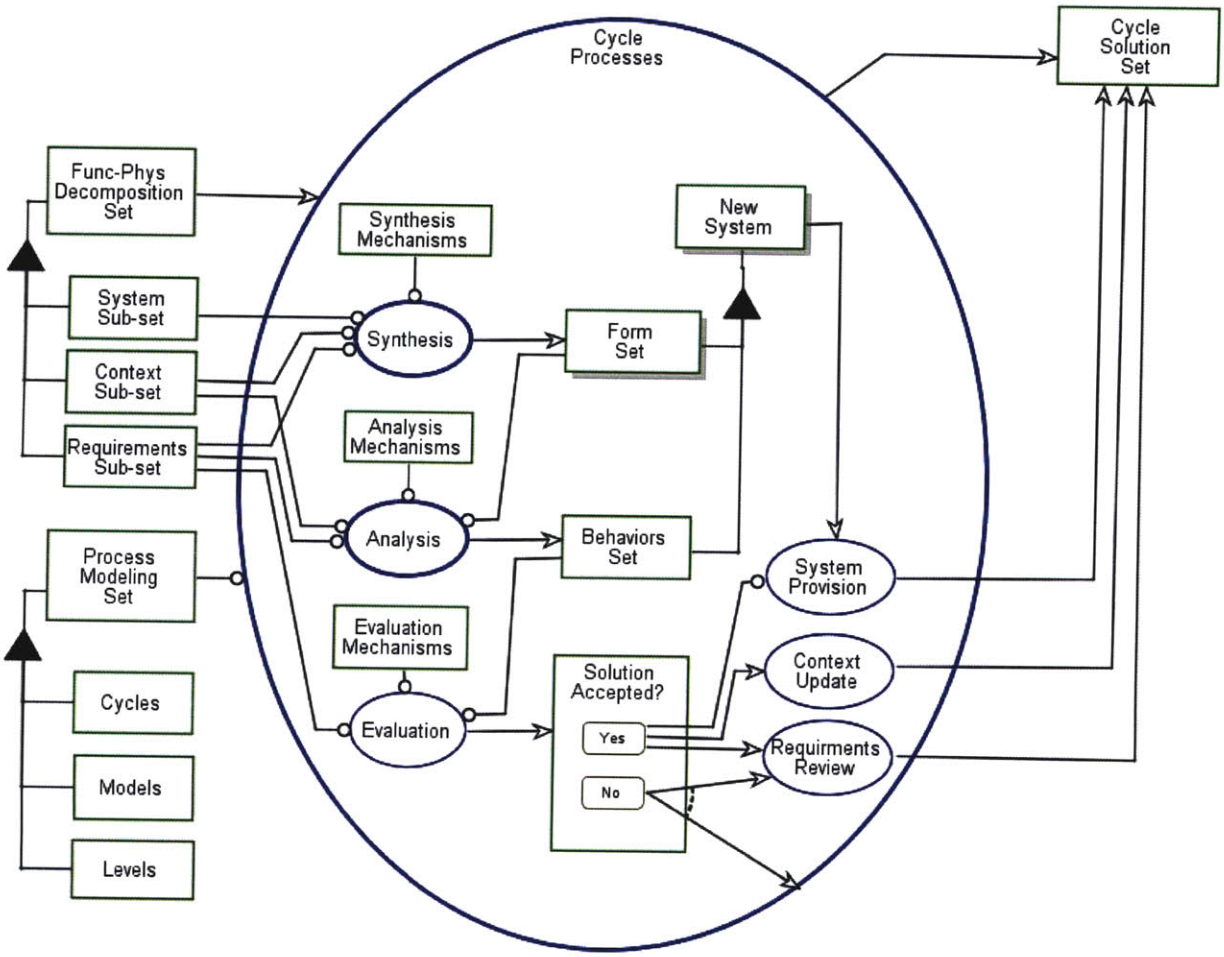


Figure 5.4:

Design cycle diagram

5.3.1.3 Design Models

A certain design cycle, will comprise several design activity models. Several of these activity models will be based on the previous decomposition stage.

These models include models for generating one or more of the system's physical configurations that should lie within the design space of the physical embodiment identified in decomposition. These types of models are known as synthesis models.

Based on the functional and behaviors decomposition, analysis models should be formulated to study the performance and behavior of the different solutions and configurations generated by synthesis models.

Design decisions about the results of the analysis models will then have to be made. This will be achieved by including evaluation models.

Finally, a mechanism for searching for the best alternative(s) will have to be devised within optimization models.

All four types of models can be mathematically modeled. Mathematical models are especially well-suited for design due to their flexibility and ease of modification. Mathematical modeling will be discussed in the following section.

5.3.2 Activity Modeling

Within this stage of the EDM we are concerned with mathematical models. These are models that can be implemented in a computer environment. We aim at building a mathematical model for each activity identified. These include all activity models of *synthesis, analysis, evaluation,* and *optimization* (figure 5.4).

Each Model has a boundary that cuts across its links to the environment defining that Model's input and output. Each Model acts like a black box transforming data from one form to another. Each Model contributes not only to the design behavior it is modeled after, but to the design process as a whole.

Domain knowledge of each discipline involved in the design informs the synthesis models to create meaningful designs and representations. The outcome of the synthesis models is

analyzed by the different discipline analysis models to predict the properties of a particular solution. The evaluation models then handle the multi-objective nature of the design. The optimization models search the design space and automate the synthesis, analysis and evaluation in search of new solutions. The process continues until the optimization has converged and a family of acceptable solutions is found.

Modeling can take place through one of two basic approaches: programming the model in a programming language such as C C++, C#, Java, etc.; or constructing it in simulation software such as CAD, FEA, or CFD. Using programming languages provides better program control and a low purchase cost. Simulation software however minimizes programming time and thus lowers project cost. The scope of any model is primarily dependent on the fidelity degree needed at a certain design level. After constructing a model, it is validated to make sure the original assumptions were acceptable.

5.3.2.1 Synthesis

The synthesis mathematical model defines the system configurations to be modeled. These models are influenced by the physical decomposition completed in earlier stages. A set of synthesis models is implemented by extracting from the design concept and decomposition, design intentions and formulating a collection of design parameters, rules or algorithms. This collection provides for a representation of the design language which in turn defines the search space. This mode of representation provides for a formalism that can be used within a computational environment to breed new designs.

The design vector or variables within it are the input to this type of Model. As discussed previously in the modeling chapter, the number and type of variables included in the design vector depends on the algorithms and structure of the

synthesis model. Synthesis Models can offer precise feedback for the EDM evolving process, on the influence of parameter variations within the design vector on geometric data.

Synthesis Models output data to analysis Models. This data includes design attributes such as dimensions, areas, volumes, locations, vectors, and mass properties for physical systems. The need for integrating synthesis and analysis Models affects to a great extent the modeling requirements for both design activities.

Synthesis models should provide for a generative mechanism. This could be done through the different techniques discussed in the modeling chapter, such as parametric and algorithmic models. Parametric models provide for a description of the artifact through parameters and relationships that allow for variation. Algorithmic models provide a description of the artifact through a set of rules and algorithms. Some examples of algorithmic models are formal Grammars. These include grammars like Shape Grammars, Graph Grammars, Lindenmayer Systems, and Cellular Automata.

The representation of generative synthesis models should encode design knowledge. The relationship between form and performance should be embedded within the representation formalism. This provides restrictions on permitted designs and ensures that the rules discard designs that do not comply with constraints. However, since synthesis models do not include performance feedback loops, it is difficult for such models to direct the generation and navigation of the search space of multi-performance design problems.

5.3.2.2 Analysis

An analysis model infers from a synthesized design solution behaviors that are relevant to a particular functional requirement. A design problem usually combines different

disciplines with each discipline developing one or more analysis models.

The outcome produced by a synthesis Model is the input to the analysis Model. These may range from simple parameters and data such as areas or volumes, to full CAD models for use in numerical analysis like FEM and CFD. The outputs of the analysis Model are performance and behavior measures that will eventually be used within an evaluation Model to assess the effectiveness of a system configuration.

In the modeling chapter several analysis models were discussed. These models range in their amount of required information input and their degree of accuracy output. Analytical models are mainly low-order (low-fidelity) models that are fairly fast but with low accuracy. On the other hand, numerical models like finite element analysis (FEA) and computational fluid dynamics (CFD) are high-order (high-fidelity) models which have higher accuracy but result in long durations which have a compound effect when such a model is run several times in a design exploration and multidisciplinary optimization process. Many low-processing approximation concepts have been utilized to generate surrogate behavior models to replace expensive and detailed analysis and simulation software when testing numerous scenarios with various input parameters (Koch et al., 2002; Bletzinger and Lähr, 2006).

In choosing a model the designer must select the best compromise between the demand for simplification and the necessity to clearly identify, describe and rate the targeted physical mechanism. A trade-off will have to be made between fidelity and analysis time.

5.3.2.3 Evaluation

The need for the evaluation of results arises while observing

systems in multidisciplinary contexts. Evaluation Models are in essence decision-making tools. The input to an evaluation Model is the output of several analysis Models. Evaluation therefore refers to the overall result of a design analysis, which encompasses multiple analysis computations.

The output of the evaluation Model depends on the strategy used in the evaluation and whether the evaluation is done before or after optimization. An evaluation is usually performed by means of an objective function which consists of a figure of merit describing the quality of a design solution. The formulation of the objective function is vital to the outcome of the design space search. A solution is expressed in an n-dimensional design space. "n" relies directly on the number of design objectives. Results from the evaluation Model usually yield a dimensionless quantity known as the quality for each solution.

In order to make a decision about rationally choosing one of the alternatives, a criterion is required which assesses all alternatives and ranks them in a certain way. The criterion, which is called the objective of the model, cannot be unique, as its choice is usually affected by several factors. These factors include the design application, timing, point of view, and designer judgment and may change with time. (Papalambros and Wilde, 2000).

In single objective optimization, the search direction can be well defined and a single solution, if exists, could be found. However, in the real world, system design problems are usually too complex and ill-defined and have several possibly contradicting objectives. This implies that there is no single optimal solution but rather a whole set of possible solutions of equivalent quality. In this set, each objective is optimized with the understanding that if any further optimization is attempted, the other objectives could be affected as a consequence. Therefore, decisions need to be taken in the

presence of trade-offs between conflicting objectives.

Addressing multiple objective problems may require techniques that are different from standard single objective optimization methods. This evaluation of multiple objectives is articulated based on the decision-maker's preferences either before or after the search.

When the preference is expressed beforehand, the designer decides how to aggregate different conflicting objectives into a single objective function before the actual search is performed. A commonly adopted approach is scalarization which consists of combining several objectives into one scalar cost function. There are different scalarization methods, such as the weighted-sum approach and the utility function method among others.

When search is performed before decision-making, the search is performed with multiple objectives at the same time. The solution space becomes partially ordered with a set of optimal trade-offs between the conflicting objectives. This set is called the Pareto optimal set.

5.3.2.4 Optimization

The final step in the design cycle involves optimizing the design to investigate the performance benefit increase. Many configurations can basically meet similar design goals. Thus an optimization problem can be put forward in order to search for better configurations. Each configuration has its individual group of design variables and functions. This implies that a design can be changed to provide various alternatives (Papalambros and Wilde, 2000).

The goal of optimization studies in this context involves studying how a design performs and how this performance can be influenced in order to choose the most desirable alternative

or alternatives (Bletzinger and Lähr, 2006).

Optimization Models are search space search machines. Searching the search space entails finding the best solution(s) within a domain of feasible solutions. The choice of an appropriate search algorithm depends on several factors, including the design synthesis model, the nature of the analysis models, the number of design variables, the existence of constraints, and the linearity of either the design variables or constraints.

The input to the optimization Model is an objective function that depends on a number of continuous or discrete values. The optimization Model seeks to minimize or maximize an objective function by varying the values of those variables within an allowed domain. The outputs of the optimization Model are new values for the design vector variables.

As discussed earlier optimization algorithms could be divided into discrete numerical optimization techniques or heuristic algorithms. Some numerical optimization techniques that handle constraints include the simplex method, sequential quadratic programming, and the exterior and interior penalty methods among others. Discrete numerical optimization techniques that handle unconstrained problems are generally gradient-based algorithms. These include Newton's method, steepest descent, and conjugate gradient among others. Within the interconnected and highly nonlinear nature of multidisciplinary design problems, it cannot be supposed that a given solution is globally optimal merely because it may be locally optimal (Atherton, 2002). Conventional gradient-based methods may not be suitable for this purpose, since they locate the optimum solution according to the point in the function space at which they started. On the other hand, heuristic algorithms are generally non-gradient methods, like evolutionary algorithms, simulated annealing, and tabu search, can escape local optima. However, no existing optimization

technique is guaranteed to find the global optimum of a nonlinear, non-convex problem.

Gradient-based methods find local optima with high reliability but might not escape a local optimum. Heuristic algorithms might find a good solution, but its optimality cannot be guaranteed since they often tend to find a different design each time they are run. In addition, they do not converge to a solution in the same effective manner as gradient-based methods do.

Furthermore, no single optimization technique is applicable in general to all types of engineering design problems. Studies in the field of nonlinear constrained problems, which are common in complex engineering design problems, have demonstrated that no single optimization technique performs best for the majority of design problems.

For a given design problem, a combination of techniques often performs better than single techniques. Using the two dissimilar methods in a complementary way creates a 'hybrid' optimization strategy that can address the problem efficiently. This strategy would ideally promote relative strengths of both methods and restrain their weaknesses in order to provide maximum analytical benefits. A heuristic technique, for example, can be applied to a problem with a high degree of nonlinearity and multiple predicted local optima to globally identify within the design space regions where best solutions may lie. Starting from the solution or solutions obtained in this exploratory search, a numerical technique can then be applied to search locally for the best solution in this specified region of interest, or also to fine-tune it. The most effective way however to solve a given problem will always be dependent on the specifics and details of that unique problem (Koch et al., 2002).

There still remain some issues when novice users apply

optimization techniques in complex design problems. These include choice of the starting point, the number of system analyses required for optimization, uncertainties in problem formulation and design parameters, and their effects on the optimization (Koch et al., 2002).

5.4 Design Integration

The term "integration", similar to the term "system", although widely used, has a lot of connotations which may carry some ambiguity for the listener or reader. The Webster online dictionary defines integration as the "act of combining parts into an integral whole". The key notion of integration is the assembly and combination of individual parts into a whole that collectively satisfies all the functional and operational requirements which would not be achieved by its subsets alone (Grady, 1994).

Formulation usually lies at the front end of the design evolving process, while integration lies at the tail end. Integration is the materialization of the modeling processes.

Like other phases within the design evolving process, integration can occur at different levels. Here we are interested in integrating models, cycles and levels to produce the specifications for a new system design state (figure 5.5).

Integration works mainly on unifying the evolving design into one whole. The success of integration lies, however, in the correctness, precision and coordination of modeling activities at each level.

Integration is primarily a bottom-up process that comprises integrating the most basic models of the system and verifying that these lower level models meet the sets of requirements, that were developed for them during formulation.

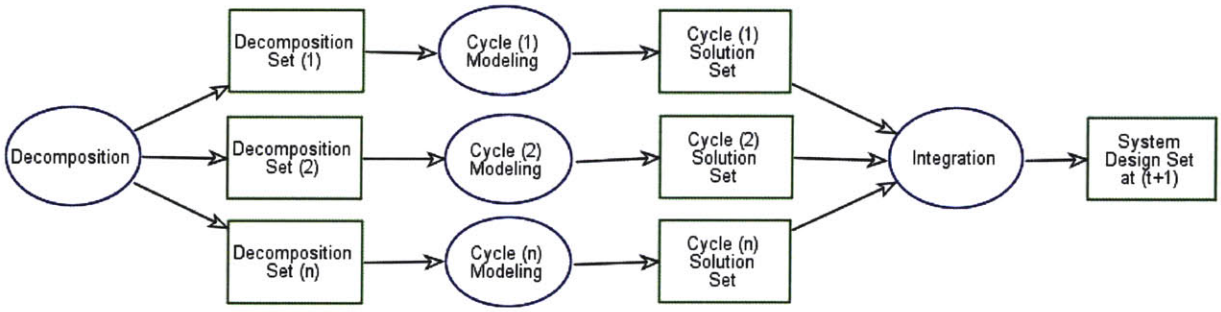


Figure 5.5:

Design Integration Diagram

Integrating activity models enables their design specialists to exchange and discuss design information, alterations in design tasks and design decisions with other specialists.

Designing interfaces is an essential part of integration. Interfaces describe the group of services that a model can provide (Pahng et al., 1997). They demonstrate detailed descriptions of how different models interact together. This includes how the models fit together, connect, and communicate. If these interfaces are compatible, activity models can consequently interact with each other.

The integration between different mathematical models in a computational environment can be carried out using integration technologies such as middleware, web services or a combination of both. The component-assembly approach in the software industry and the recent focus on component-ware can also benefit in assembling interacting activity models.

The system architect decides on the data that will be shared from one model to other models. This data should pass between models in an automated fashion as soon as all models are linked together.

Managing dataflow from one model to other models has always been extremely time-consuming. This can be overcome

by providing execution scheduling functionality and easing activity model communication. Design information that a certain activity model desires to receive represents that models interests. These interests could trigger the action of receiving the well-suited design information as soon as it is generated by another model. Implementing automation could minimize the time required to run design iterations.

Solution coordination is an important factor in achieving a solution to the design problem through multiple solutions for the different levels and cycles.

There are certain risks that the integrated models may not actually work as formerly planned. Testing the system involves running the simulations and reviewing the model validity. With increased experience, system architects can predict more of these risky and negative interactions until the integration task becomes much easier.

At the integration phase the design process that started at a certain design state with the formulation of a set of requirements comes to an end. This stresses the need for Requirements management throughout the process.

Requirements Management is defined as "the identification, derivation, allocation, and control in a consistent, traceable, correlatable, verifiable manner of all the system functions, attributes, interfaces, and verification methods that a system must meet including customer, derived (internal), and specialty engineering needs." (Stevens and Martin, 1995) Requirements management addresses which requirements have been changed when and by whom; and how to trace each requirement; to which components has each requirement been allocated.

6. Managing EDM Complexity

One of huge challenges in developing the EDM is the issue of design complexity Management. In the design of complex systems designers face significant challenges during the design development, including:

- Design knowledge content and resolution variation
- Design decisions are coupled and involve tradeoffs
- Design decisions are made under uncertainty

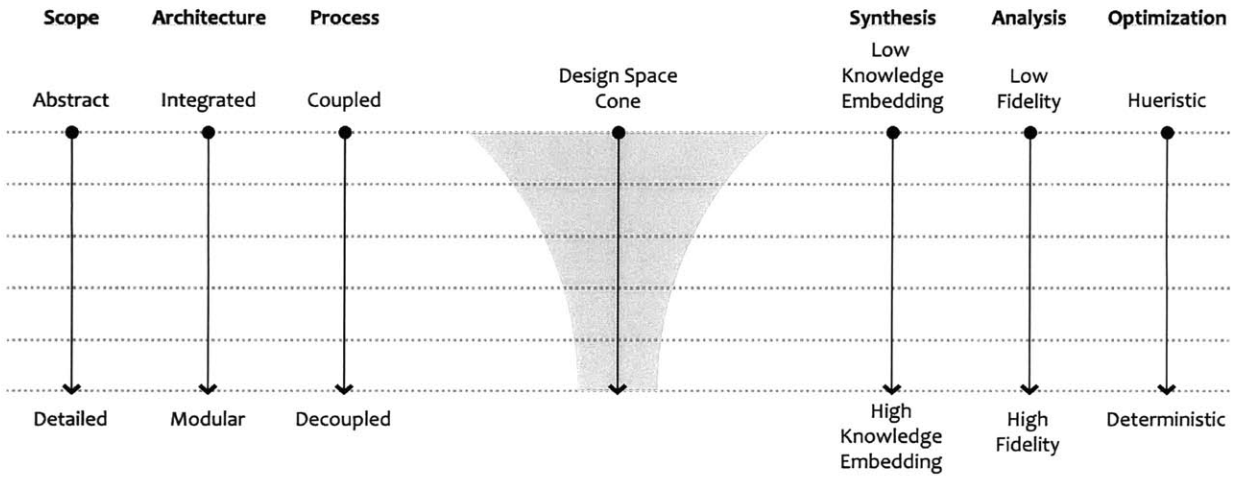


Figure 6.1:

Design development along the EDM life cycle

Because of these challenges, designers often have to backtrack in the design process. They may have to reconsider earlier decisions when new information is revealed. This iteration is undesirable because it costs time and money (Aughenbaugh and Paredis, 2004).. The nature of these challenges is therefore important to improving the design process. Designers need to recognize the ways in which knowledge is acquired and the

fidelity level needed at any stage, the coupling of different design decisions, and the types of uncertainty under which these decisions are made

6.1 Resolution

Throughout the progression of design evolution, there is an inherent relationship that is the core of all design development processes: the inverse relationship between design knowledge and freedom. As the design evolves, design freedom rapidly decays while knowledge about the design object continuously increases. As the process moves forward, designers gain knowledge but lose freedom to act on that knowledge (figure 6.1).

Designers move from simple and generic designs into more complex and detailed ones throughout the design process. Early on in the process, the exact structure of design objects is not clearly defined (Rosenman and Simoff, 2001). With project progress, the design description must evolve and change, as well as the constraints and synthesis and analysis models. Practically, the level of description of a specific design should be directly proportional to the amount of information available at a specific design state. A design could not be described at the fabrication level when the project is still at an early state, as too much information impedes the design's progress.

As the design evolves, designers can estimate the effects that possible decisions have on the systems behaviors using modeling and simulation. Designers can use these estimates to make better decisions and to design a system that is more likely to meet its functional requirements. Although the general system behaviors already have been estimated at a higher level, with each decomposition decision, the estimates can be improved (Aughenbaugh and Paredis, 2004).

Design description complexity and mathematical model

sophistication and resolution increase as more detail is added, moving from simple representations to more detailed descriptions.

In early design states, designers can synthesize many alternatives, and pertinent analysis can be conducted. In later states, however, more detail and resolution is required to perform elaborate synthesis and analysis. These are conducted using higher-fidelity modeling and tools. A simple system such as one that incorporates beam representations of structures can be easily modeled. But, when it is substituted by a plate or finite element model, the number of design degrees of freedom and system dimensionality increase remarkably. With this evolution, higher-fidelity analysis is often required (McManus et al. 2004).

For the EDM implementation, models with different resolutions and granularity levels are needed. By altering models or exchanging existing disciplinary synthesis and analysis models for more suitable fidelity levels, existing EDM level models can be evolved to lower successive levels.

Furthermore, the nature of the design problem itself can change with design progress. In emergent situations, initial design vectors, parameters and models may become irrelevant. In order to move forward with identifying solutions and exploring design spaces, relevant models have to be identified and instantiated. This involves dealing with more complex design parameters and results, which increase computation time, making the enhancement of the fidelity of disciplinary analyses a difficult task (McManus et al. 2004).

Varying model resolution can be implemented in two directions: vertically and horizontally. Vertical variation of resolution takes place between the different levels of the EDM evolving process. On the other hand, horizontal resolution variation can occur within one design cycle. For example, two

models could model the same behavior with one model running at a higher-fidelity level, and therefore taking a longer time to run, while the lower-fidelity model runs faster but does not provide accurate answers. In this case, the system architect could use the faster low fidelity model within the optimization, and at different intervals of the optimization verify its results using the higher fidelity model.

In addition, multi-resolution representations will have various modeling needs that can intensify the design challenge. The primary concern of multi-resolution modeling is resolving representational discrepancies that exist between models (Davis and Bigelow, 2002). Having different models working at different levels or within design cycles implies the need to preserve consistency at each abstraction level. Reynolds et al. (1997) discuss the challenges in this process. Design strategies that take these potential discrepancies into consideration are necessary for designing these cross-resolution models.

6.2 Coupling

Designers have to deal with coupling between behaviors. Designers need to understand the relationship between different behaviors in order to take decisions. The interaction between the functions and subsystem results in a coupling of the design decisions. In some cases the coupling between two behaviors can be beneficial because it allows synergies that improve the overall system. In order to take advantage of such opportunities, designers need to understand the tradeoffs available at design time. Modeling and simulation can support this process (Aughenbaugh and Paredis, 2004).

Decoupling takes place when the interactions between parts of the system disappear. This happens when the various interconnected behaviors and models are partitioned into subgroups which do not require the output of another group as their input. Doing so usually minimizes the degree and risk

of failure in any one part of a system if another part is altered. The system structure is thus simplified and can benefit from parallelism.

Coupling or dependency within the context of models is defined as the degree to which each model relies on each one of the other models in a given system (Kroo, 1997a). Low coupling, or "loose" or "weak", denotes a relationship where one model interacts with another one through a stable interface without any concern about the internal implementation of the other model. Thus a change in one model does not require a change in the implementation of the other one.

High coupling, or "tight" or "strong", occurs if one model changes or relies on the internal implementation of another model, such as accessing local data from it, and so the dependent model will change according to manipulations in the way the other model produces data. This is also known as content coupling (Kroo, 1997a).

This notion of independence and interdependence of models must be identified for any design. The system architecture must allow for both the independence of structure and the integration of function.

Modularity is an example of decoupling. Modularity is a specific design structure where parameters and tasks are interdependent within models and independent across them. Modules in a larger system work together as units but are structurally independent of one another. This implies that a module's internal structural elements are strongly linked among themselves but weakly linked, with gradations of modularity to elements in other modules (Baldwin and Clark, 2000).

Decoupling can also happen between successive levels within

an EDM as the system evolves (figure 6.2). As the design of the system progresses, physical parts and components and their associated functions that are weakly related and have very little influence on other components and aspects can be synthesized and analyzed individually.

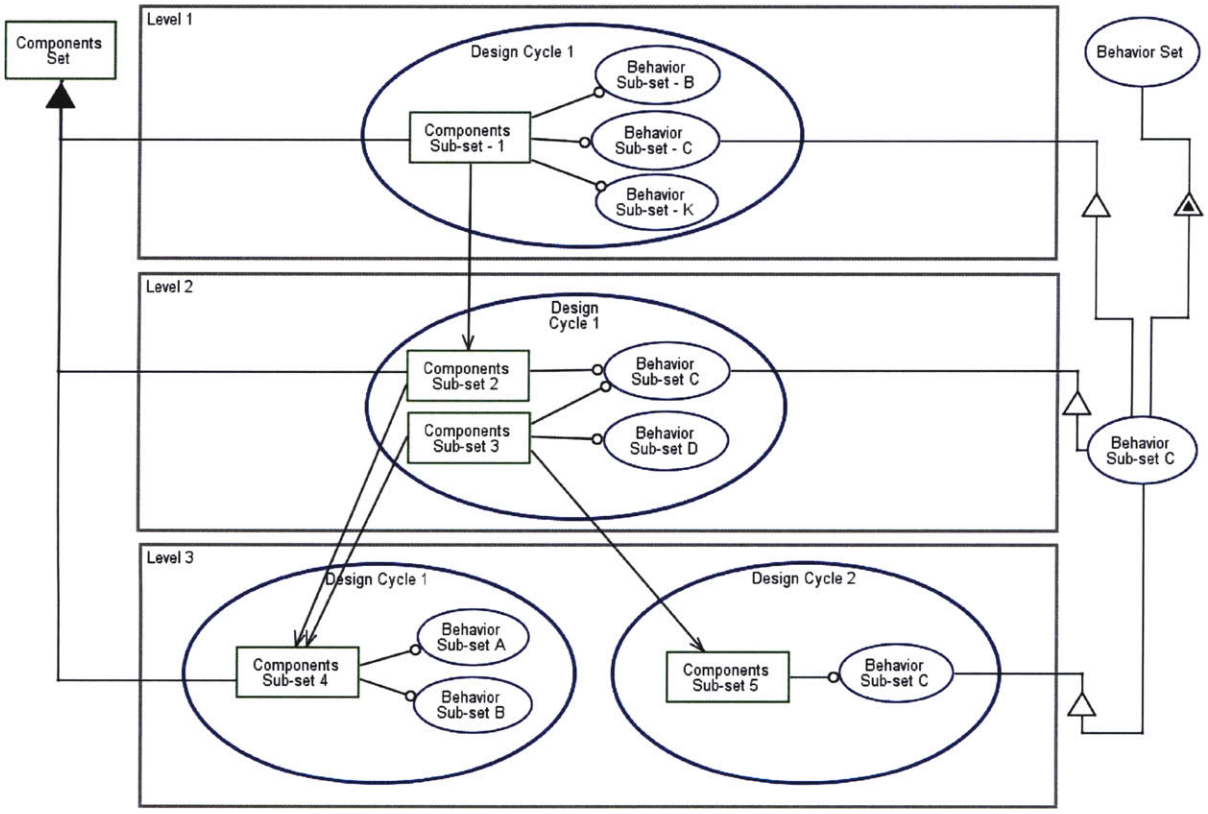


Figure 6.2:

Decoupling can also happen between successive levels within an EDM as the system evolves.

A design cycle within a specific level that is intended to generate certain system component configurations can therefore evolve in subsequent lower levels of the EDM into two or more decoupled design cycles. Furthermore, new cycles and new models may be created as new levels surface in the EDM. Therefore, the EDM architecture is expected to be integrated in higher design levels and modular in the lower levels.

5.2.3.3 Uncertainty

Uncertainty is mirrored in design. Designers try to answer questions on system's requirements, operational context and environment, future design decisions and possible emergent behaviors. Answers are typically uncertain. Understanding uncertainty helps designers make rational decisions, and eventually find better answers.

Uncertainty is of two types: aleatory uncertainty and epistemic uncertainty (Parry, 1996). Aleatory uncertainty is: "a potential deviation from reality in a prediction or model due to natural random (stochastic) behavior" (Aughenbaugh and Paredis, 2004). Due to its inherent randonmess, aleatory uncertainty can be represented via classical probability theory. Aleatory uncertainty is exhibited in phenomena driven by stochastic processes such as machining errors, materials variations, and communication systems errors, among others.

Epistemic uncertainty on the other hand is caused by the lack of knowledge of reality. In a model, epistemic uncertainty can generate deviating results in terms of depicting behavioral changes in reality. Epistemic uncertainty is also called imprecision, ignorance, reducible, or subjective uncertainty (Aughenbaugh and Paredis, 2004).

Epistemic uncertainty should not be represented using probability density functions, this is due to the lack of information describing the likelihoods of events taking place,

The presence of epistemic uncertainty limits the EDM processes. The ultimate success of a decision is highly affected by uncertain factors. Thus, without a means to estimate, study, control or reduce uncertainty, designers decisions cannot account for factors that can negatively affect the expected consequences of their decisions. (Aughenbaugh and Paredis, 2004).

When making decisions, designers usually try to gather more information, or wait until it becomes available when the wait is feasible. However, designers often face situations where they must make decisions in order for the design process to progress. In such situations, designers tend to backtrack while evolving and building their systems to account for possible errors in requirements. Such errors generate for two reasons: requirements defined in higher hierarchy levels are overly restrictive or unexpected interactions among system's components (subsystems) due to uncertainty. The first reason highlights designers' lack of information about feasibility, and the second highlights their lack of information about subsystems behaviors (Aughenbaugh and Paredis, 2004).

It is important to note that most system-level behaviors cannot be known before the whole system is fully designed. To make progress, designers tend to use simulation methods to better understand the implications of their decisions in regards to defining the search space, or selection of system's behaviors, etc. Simulation can be used in all design stages to reduce and define uncertainty in models for systems' behaviors. That being said, simulation cannot fully eliminate uncertainty from models, thus backtracking is always expected whenever additional information is revealed (Aughenbaugh and Paredis, 2004).

7. MASDAR City EDM

7.1 Background

Masdar City (Arabic مصدر, masdar, *the source*) is a city being constructed close to the city of Abu Dhabi in the United Arab Emirates. It was designed by the British architectural firm Foster + Partners and will be a sustainable, zero-carbon, zero-waste ecology that will depend on renewable energy sources. The project will take about eight years to build with an estimated cost of US$22 billion.

Figure 7.1:

Masdar City Master Plan (Foster, 2007)

It was initiated in 2006 and is expected to be habitable in 2009 (Locke, 2008). A university will also be founded in the city, the Masdar Institute of Science and Technology (MIST) which will be supported by the Massachusetts Institute of Technology (MIT).

Masder city will join a small number of planned cities and science developments like Tsukuba Science City in Japan, Novosibirsk's Akademgorodok in Russia, and the Education City in Qatar. These cities are specialized research and technology intensive municipalities that include a living environment.

7.2 Masdar EDM

The EDM presented here is part of an ongoing research project. The work completed so far includes the design state description in addition to the formulation, decomposition, and several modeling phases within the design evolving process. Modeling will only include the synthesis and analysis models of one design cycle within the top design level. The research should continue the following year to include evaluation and optimization models that will complete the design cycle, as well as other design cycles within the current level and other lower design levels.

7.2.1 Masdar Design State

Based on the description of a design state mentioned earlier, the Masdar design state should include three sets: a requirements, context, and system design sets.

7.2.1.1 Requirements

At the time this thesis was written we were unable to obtain the official Request for Proposal (RFP) for the Masdar project that includes the original stakeholder's requirements.

Therefore, identifying the full scope of the functional requirements and the stakeholders' needs will not be possible.

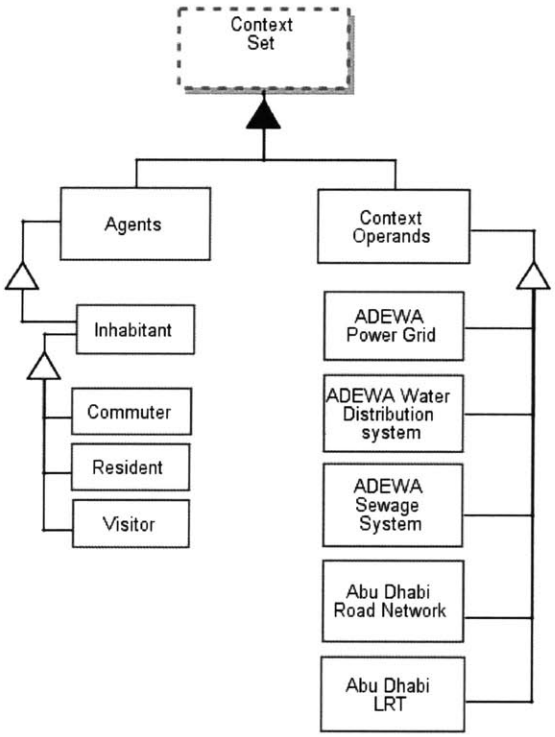
7.2.1.2 Context

Masdar City is being constructed 17 kilometers (11 mi) east-south-east of the city of Abu Dhabi in the United Arab Emirates. Masdar city will have an area of 6 square kilometers (2.3 sq mi) and will house 50,000 people and 1,500 businesses comprising mostly of commercial and manufacturing facilities that specialize in environmentally-friendly products.

More than 60,000 workers are expected to commute to the city daily (Locke, 2008). The city will interface with existing systems including ADEWA power grid, ADEWA water distribution system, ADEWA sewage system, the Abu Dhabi transportation network which includes the road network and the Light Rail Train (LRT) as illustrated in figure 7.2.

Figure 7.2:

Masdar City Context



7.2.1.3 System Design Set

In this experiment, we will focus in the system design set on the super system level that consists of the city level as illustrated in figure 7.3 (de Weck et al. , 2009).

Within the system's Form set five sub-systems were considered: The Building System, Transportation System, Energy System, Water System, and Waste System.

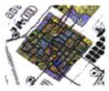
These sub-systems have several associated behaviors That include Sheltering, Transporting, Energy operations, Water operations, and Waste operations as well as cost.

As illustrated in figure 7.3 several dependencies exist between the different sub-systems and sub-behaviors which have to be considered when developing the Masdar EDM. In the rest of this section I will describe some of the subsystems of Masdar city in more detail.

Building System

Based on the executive summary presented by Foster and Partners the main factors in the design of the master plan are the provision of area requirements which include a university located within a special economic zone with associated commercial, light industrial zones and residential accommodation, all of which should be established within a context of a carbon neutral zero waste city.

The requirements also require a design with a high quality of urban spaces and buildings within which a sense of place is created and through which a community will prosper and a pleasant environment will be formed making the city a place where people would want to live in.

System Layer	Specific Examples	
Super-System Level ("City" Layer)	Overall City Layout, Transportation Architecture and Interfaces, Zoning, Personal Rapid Transit (PRT) Topology	
System Level ("Block Layer")	Design of local water, energy and information Systems, Distribution of Cargo Systems, Software and Information Systems	
Sub-System Level ("Building Layer")	Efficient Building design, renewable energy source integration (photo-voltaic), smart building skins, PRT vehicle design	

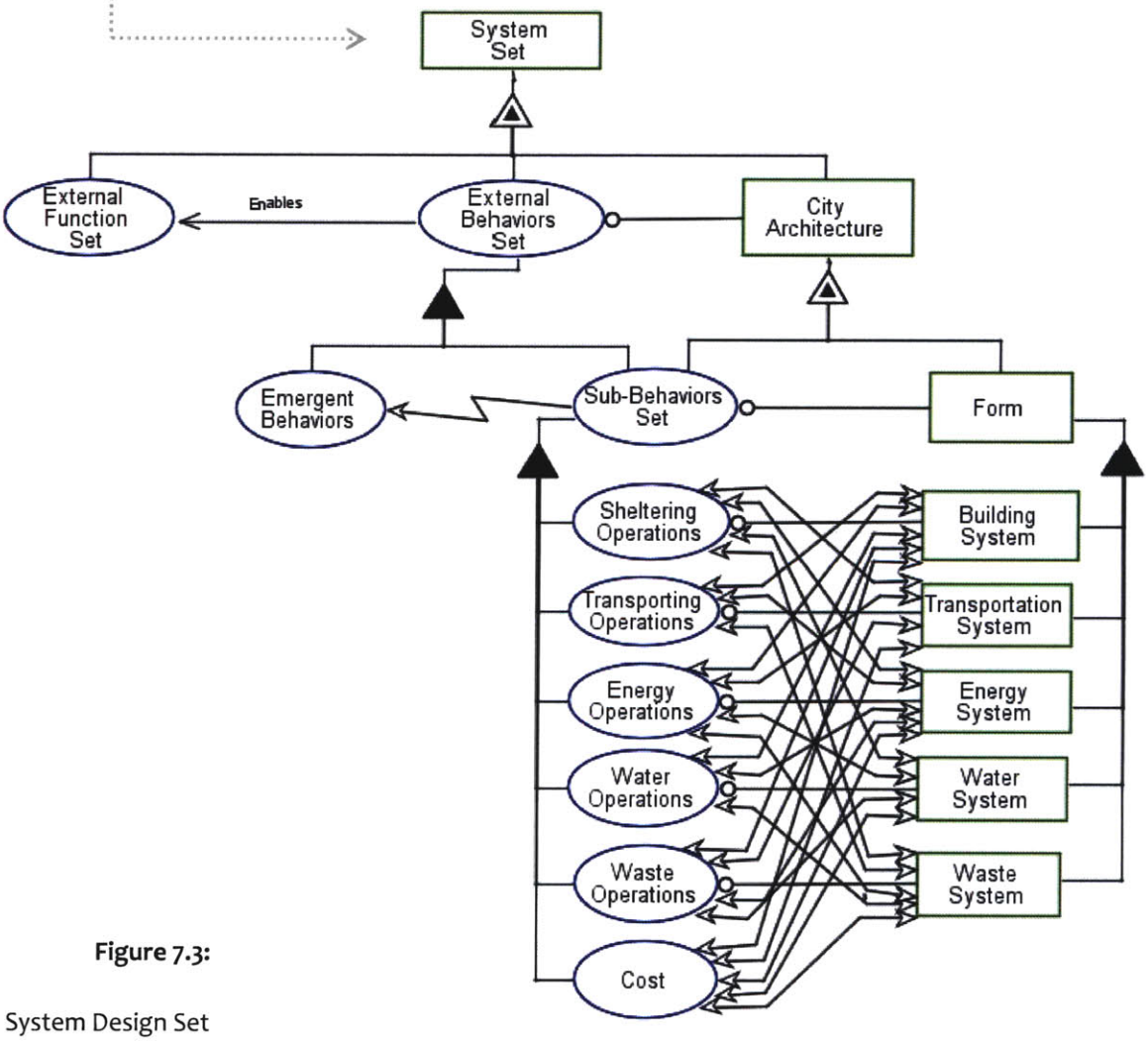


Figure 7.3:

System Design Set

Foster states that their design is based on the principles of sustainable urban design that include aspects such as, low rise buildings, high density accommodation, sustainable transportation, dense neighborhoods and controlled spread of development (Foster, 2007).

The Foster design team believes that dependence on the Energy Systems to produce sufficient energy is not enough to support the demands of the city. Energy load reduction strategies have to be implemented. Initial energy analysis carried out by the design team stated that sufficient energy cannot be produced to meet demand without a 30% decrease in demand. By implementing sustainable design strategies within the urban and building system such a load reduction can be achieved (Foster, 2007).

These strategies include reducing cooling loads by shading external walls and roofs similar to old Arabic cities. Historically, the walls were designed with restrictions on the number and size of openings especially in walls exposed to direct sunlight. Furthermore, the walls were constructed of dense materials and have a width and mass which absorbs heat during the day and releases it at night, reducing thermal gain. In addition, access to residential buildings was through private courtyards which had water features, planting and shading that help reduce temperatures locally (Foster, 2007).

Such strategies among others were incorporated in the design of the building system by the Foster team. A study of street widths and proportions was carried out to verify optimal shading strategies. Traditional elements were also incorporated into the design of the individual typologies to provide an appropriate scale and grain to the city. Examples of such elements are screens and balconies that shade window openings onto private courtyards. In addition, reduction in cooling loads can be achieved through the study of the orientation of buildings in relation to prevailing winds and sun

paths in accordance with proven historic layouts (Foster, 2007).

The main components of the urban and building system will be: the Central Spine, MIST, SEZ, Headquarter and Office Area, Commercial Zones, Light industry, Residential accommodation, Car parking, and Green Fingers. A brief description of these zones will follow.

Spine

The key design concept of the urban and building system is based on a central spine design patter with different city zones relating to it. This central spine will run throughout the city in line with the route of LRT and along it all primary centers will be located. These will include the culture center, MIST, the Friday mosque, and hotels. The spine will help create an orientating device inside the city also giving access to the functions by LRT and PRT (Foster, 2007).

MIST

The Masdar Institute of Science and Technology (MIST) is the first building to be constructed and is also the main building of the city. It is located on the central spine near an LRT station and will be the focus of the different zones and uses that are distributed through the city.

The Institute will create a center of quality with its associated Special Economic Zone and research and development areas. MIST will serve as an attractor for their facilities which in turn establish the extent of the commercial and light industrial component (Foster, 2007).

SEZ and Office Areas

The commercial districts and SEZ are set around the University

and along the Spine. This will help attract research and development companies. The city planners identified the commercial center as a "permitted increased height zone" which will create increased density (Foster, 2007).

Headquarters

The office building for the Abu Dhabi Future Energy Company (ADFEC) will be located on the central spine and close to MIST and the SEZ and will be well connected with transport links (Foster, 2007).

Commercial spaces

Commercial spaces will be located mainly on the central spine where high end retail outlets and top class restaurants will be found. In addition commercial spaces will be located in city squares and will contain restaurants cafes and food outlets providing essential community services. This is intended to play an important role in the liveliness of the city providing residents with premiere services and therefore reducing their need to travel far from home for individual needs. Additional family services, civic and community buildings are also located in this area (Foster, 2007).

Residential accommodation

Residential accommodation includes many types and sizes of apartments, duplex apartments, town houses and one, two, three and four bed units. Residential accommodation is mixed with other uses throughout the city to guarantee an active environment throughout the city. Schools, mosques, hotels, will be spread throughout the residential zone to create events throughout the zone and will ensure that the zone does not become wholly residential. In this way the city districts are enlivened. The mosques and schools create centers for local communities giving them a sense of place. (Foster, 2007).

Car parking

Parking for the residents is provided in seven car parks located in the wall zone. They will have an average height of 3 stories with the space above the wall occupied by residential purposes at about 2 stories high. Parking spaces are allocated on the basis of the shortest distance from the Residential unit to the car park (Foster, 2007).

Light industry

Located on the perimeter of the city in the "wall" zone are the light industries. This location helps with access and deliveries by heavier vehicles without compromising the city's transportation infrastructure (Foster, 2007).

Public open space

Masdar city will have two forms of public open space: green fingers and public squares. There will be two green fingers with a combined area of over 20 ha. In addition, the public squares will be each over 1,100 m^2 in average. The squares and green fingers should be easily accessible and will serve as reference points on the mental map for the residents of the city and will stand in contrast to the tight urban grid. (Foster, 2007).

Transportation

Given that automobiles will be banned within the city a coordinated approach between external public transport interface and internal mobility systems was developed by the design team.

Internal transportation will be accomplished through the Personal Rapid Transit (PRT) systems, which will be located based on population density movement throughout the city.

The PRT system will be accommodated within the undercroft of the city which is designed to also house the infrastructure distribution. Therefore, the undercroft will become a fully integrated transport and services distribution network

Masdar will connect to other locations outside the city through the Light Rail Train (LRT). There will be four stations within the city and will be connected outside the city to the airport, Yas Island and Raha Beach. The stop at ADFEC headquarters within Masdar will be the key node for public transport. The LRT will be elevated throughout the city system and will run through the site from north to south.

The density of the network required to provide the Masdar city residents with a high quality service is such that it is essential that it could not be accommodated at pedestrian level, or be elevated above the streets. To satisfy walking requirements, 50% of users should not need to walk more than 100m to any station and the maximum distance allowed will be 150m (Foster, 2007).

Energy

In addition to the design strategies implemented in other systems such as increased insulation, improved envelope permeability of buildings, and sustainable transportation modes; the city will need to have an efficient generation of energy to meet the city's demand.

The essential three steps identified by the design team to maintain development are:

- Load Reduction and passive design strategies in different systems
- The use of renewable energy resources.
- Optimization of supply systems.

The design team believes that the renewable energy resources that will be proposed must work with proven obtainable technology to ensure the success of the city.

Masdar will employ a variety of renewable power sources that will largely incorporate a series of solar technologies such as Concentrating Solar Power (CPS) and Photovoltaics (PV). PVs will be placed on rooftops to provide supplemental solar energy. A variety of cell types are proposed to allow an evaluation of performance and deny dependence on only one type of cell

In addition wind farms will be established outside the city's perimeter. Furthermore, evacuated thermal collectors, concentrated thermal power waste to energy generation as well as Geothermal energy will be investigated (Foster, 2007).

The design team hopes that the city will act through its design as a technological incubator and test bed to incorporate emerging energy technologies.

Water

Water supply is a resource similar to energy that must be considered in the perspective of reduced demand. The Foster team believes that the current expenditure of 350L per day in Abu Dhabi must be reduced to a more practical 140L per day which they anticipate to achieve through the use of competent devices and metering of supply with a sliding scale of charges.

A solar-powered desalination plant will be used to provide the city's water needs. Main water supplies will be groundwater desalination and sea water desalination.

Furthermore, 80 percent of the water used will be recycled and waste water will be reused. Efficient Gray water recycling will cover 30% of building demand and will be used for irrigation

and other purposes. (Dilworth, 2007)

Waste

The city will attempt to reduce waste to zero. The Foster team proposes that the sustainable waste management strategy follows a hierarchy of 'Reduce, Reuse, Recycle, Recover and Disposal'. (Foster, 2007).

Initially through this strategy waste will be minimized at source. Packaging will be removed at the consolidation center and recycled. Next resources will be reused for the same purposes for which it was designed and without processing which is a cost effective and convenient method that minimizes waste production. Recycling comes next in which waste is altered for a new use.

End users will separate waste into three streams and waste will be collected at the undercroft level by PRT. Segregated waste will then be further separated into paper, metals, plastics, glass and compost. Later on the recoverable component of a recyclable waste is thermally treated. Electricity and waste heat will be generated and a balance between the power generation and recycling ratio needs to be established (Foster, 2007).

Furthermore, the design team suggests that the city construction will use materials that have the least embodied energy throughout its lifecycle, which will minimize manufacturing, transport, operational, and recycling energy. In addition cost, safety implications, health and other factors will be taken into consideration. For this, the design tam believes a proper assessment of materials throughout the lifetime of a building is necessary.

7.2.2 Masdar Design Evolving Process

As described previously in this thesis the evolving process consists of four main phases: Formulation, Decomposition, Modeling, and Integration. In this experiment we will carry out the first two phases in addition to building the synthesis and analysis models. As stated at the beginning of this chapter, the rest of the models as well as the integration phase will be completed in a later stage of this research project.

7.2.2.1 Formulation

The formulation process starts by drawing the system boundary for the design problem being investigated. This will be followed by the identification of the most important design objectives. Finally a design concept should be established to help guide the following decomposition phase.

7.2.2.1.1 System Boundary

As stated earlier, one of the most important early tasks of designing a system is drawing its boundaries. Based on the executive summary presented by Foster and Partners (Foster, 2007), one of the most important requirements to fulfill in the master plan design was the establishment of the building system. Therefore, we will focus in our current evolving process on the building system. The other systems will be addressed in following evolving processes.

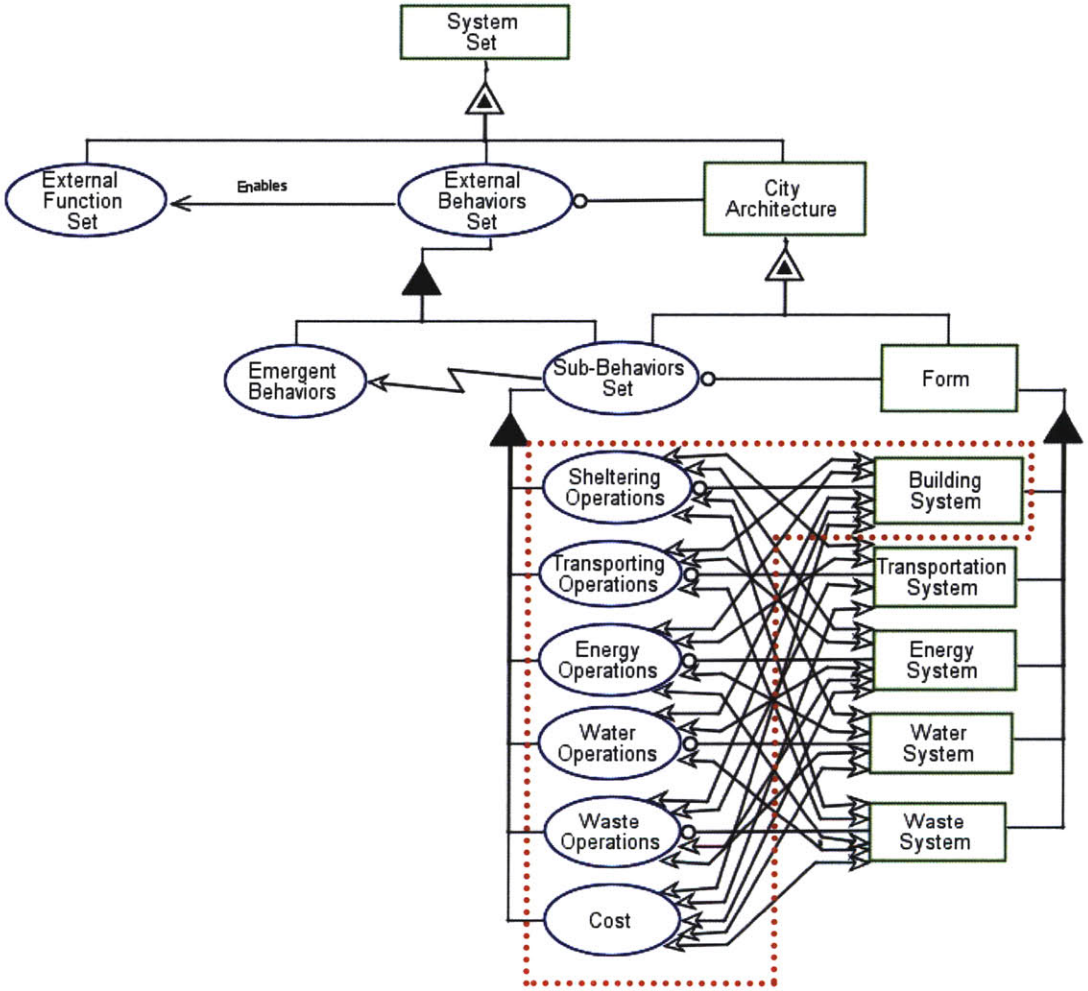
As illustrated by figure 7.4 the building system is associated with several behaviors including Sheltering, Transporting, Energy operations, Water operations, and Waste operations as well as cost.

The system boundary for the building system will be drawn following the Masdar's city walled zones. This boundary will cut across the links with the external facilities and other

elements of the city's context. This boundary will help determine the inputs to the city's building system within the walled zone, the amount of resources it will consume, as well as the outputs it will produce. Other system boundaries shall be developed for the following evolving processes in the EDM's design life cycle.

Figure 7.4:

Identifying the System boundary within the Formulation phase



7.2.2.1.2 Objectives Hierarchy

As stated earlier, based on the executive summary presented by Foster and Partners (Foster, 2007), the key objective in the master plan design was to establish the building system. This will require the provision of the area functional requirements as well as adjacency network requirements which are part of

the expected sheltering behaviors. These requirements are part of the highest priorities at this level.

The allocation of the different zones of the building system produces behaviors that can have an effect on other systems down the line such as the transportation system. Therefore another objective identified at this level was the minimization of the residents commuting time to work and green areas.

Other behaviors might be generated by the building system allocation of areas and can affect the design of the energy system. Therefore an objective was established to reduce load and maximize the use of renewable energy resources such as PV's on rooftops which are directly associated with the building system.

Other behaviors that are associated with water and waste operations are mostly either consumption or production behaviors. This means that given a certain allocation of areas within the city's walled zones the consumption of water can be established. Furthermore, given the same allocation the production of waste can also be estimated. Similarly the building system can affect the construction cost within the walled zone. This can be estimated for a certain area allocation. Therefore, given that the building system can only have an effect on either the consumption or production of these behaviors, their design priority will be lower within the current evolving process.

7.2.2.1.3 Design Concept

The master plan concept developed by the foster design team for the building system defines the city layout, distribution of uses and population densities. This is based on defining a city cell which determines the plot area, density, floor area, height and planning, population, use of Photovoltaic cover, energy loads, water demand, and waste generation (figure 7.5). This

data informs the following design phases of the transportation system, the infrastructure design of energy water and waste facilities, and cost estimates.

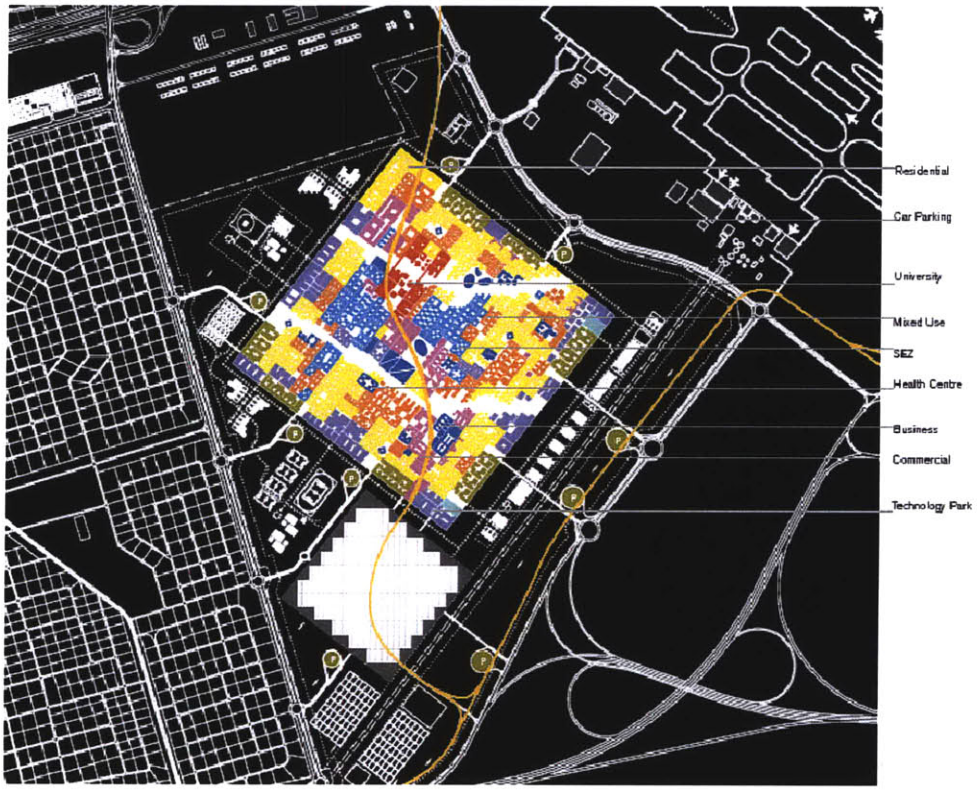


Figure 7.5:

The concept developed by the foster design team for the building system (Foster, 2007)

7.2.2.2 Decomposition

At this level both the buildings system and its associated behaviors will be decomposed. A mapping will be established between the requirements and the physical embodiment within the building systems and its associated behaviors.

The building system will decompose into the different zones. Each of the zones will have a set of attributes that include location, area, height and adjacency. With each allocation generation a set of behaviors will be generated, these include areas, commuting to work and greenery, energy consumption and energy generation by rooftop PVs, water consumption, waste generation, as well as construction cost (figure 7.6).

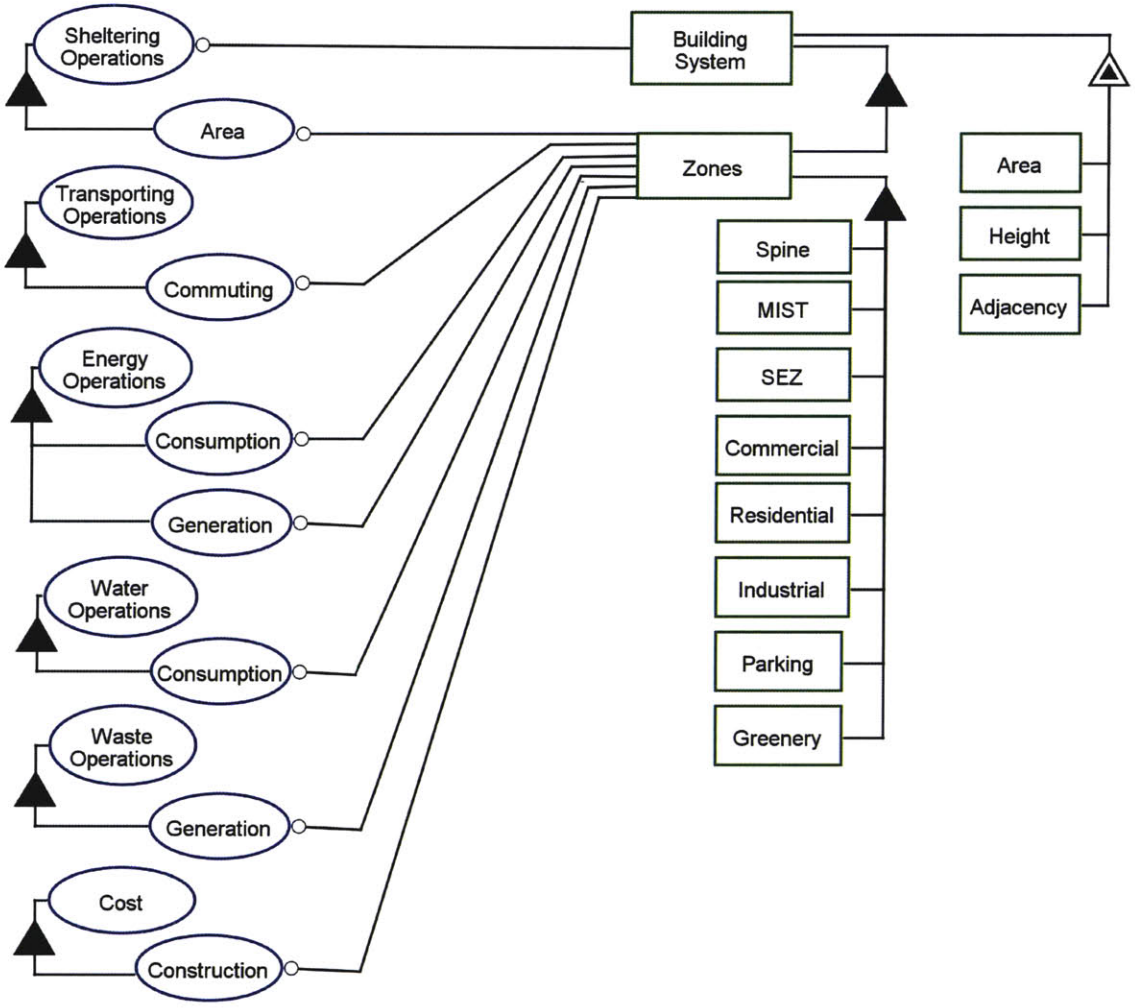


Figure 7.6:

The Physical - Functional Decomposition of the Building System and the associated Behaviors

7.2.2.3 Modeling

Process Modeling

Design Levels

As stated earlier we decided to subdivide the design problem into three levels (figure 7.3). Our initial efforts were focused on the "City level" (de Weck et al. , 2009).

We then partitioned this level into sub-levels. Based on the

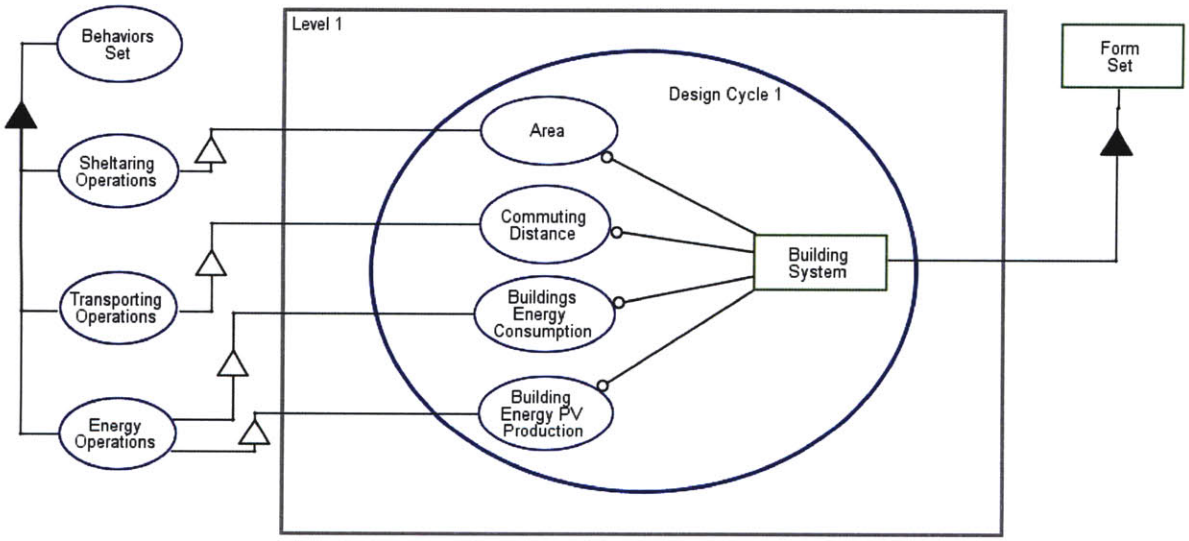
objectives hierarchy, the first level will focus on the sheltering, transporting, and energy operations sub-behaviors. The following levels will handle the water, waste and cost behaviors.

Design Cycles

Due to the coupling that exists between the different behaviors within the first design level, we decided to use a single design cycle that can help us asses the tradeoffs between the different behaviors (figure 7.7). This cycle will be able to synthesize, analyze, evaluate and optimize the layout of the building system within the city.

Figure 7.7:

The Design Cycle Diagram for Level 1



Design Models

Figure 7.5 shows the current allocation proposal for the various building zones, including the university, residential zones, commercial zones and recreational areas amongst others. This particular layout is only one amongst many that could have been chosen. Our goal is to build a computational design system capable of recreating a close approximation of the

current MASDAR city building system, but also can create alternate designs. This system will serve as a "computational laboratory" that can support ongoing decision making and maturation of MASDAR city or can be applied to future developments within the city. The synthesis and analysis models will be described in the activity modeling section that follows. The Evaluation model that we are currently working on is based on a simple weighted sum approach. The Optimization model will implement a Genetic Algorithm (GA).

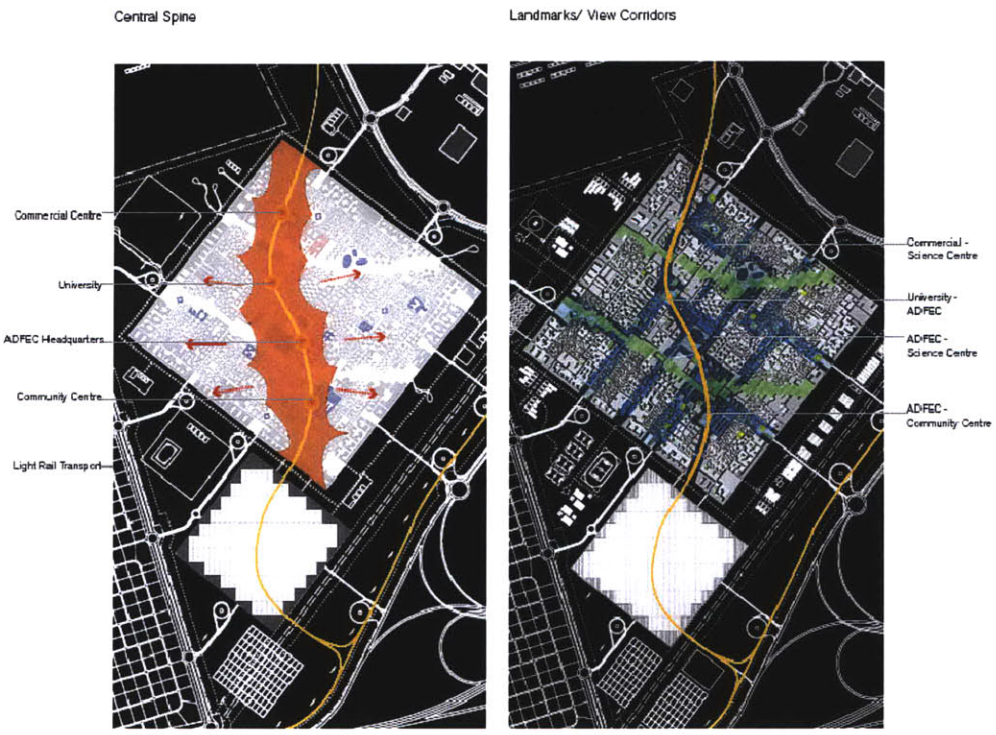
Activity Modeling

Synthesis Models

Within the Masdar master plan the dimensions of the walled city square are about 1440m x 1440m or 2,073,600sqm. The square is subdivided into a grid of 40x40 cells with each single cell having a size of 36m x 36m or 1296sqm.

Figure 7.8:

Two design patterns implemented in Masdar City: "central spine" and "Green fingers"

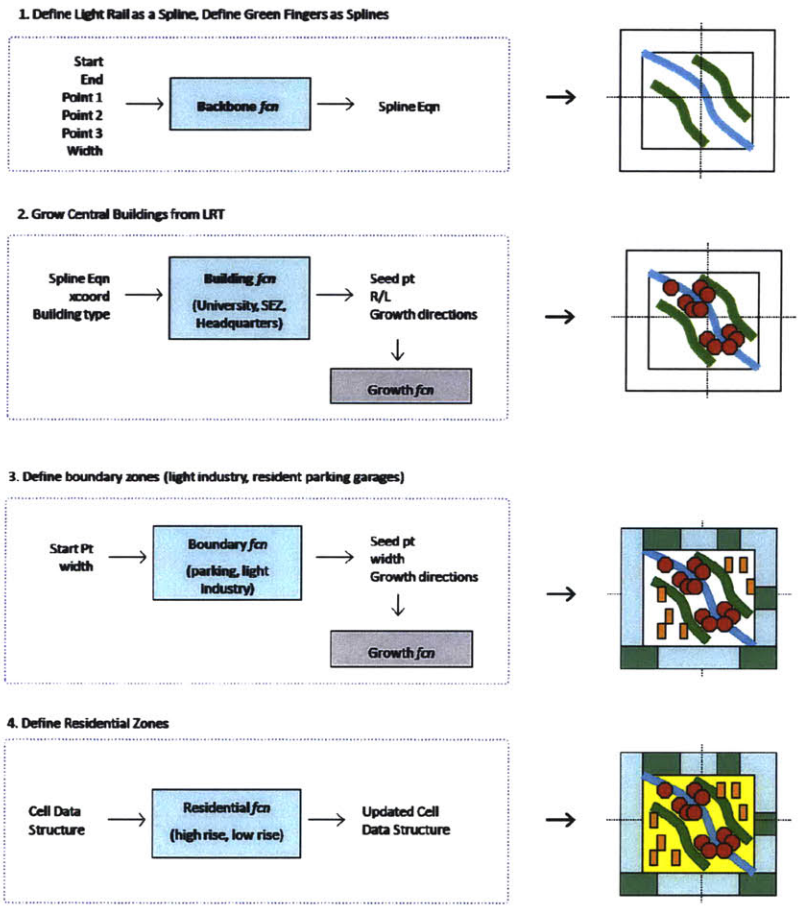


If we consider a one to one mapping of each cell attributes to a variable in the design vector we will need 1600 variables. This is a large number that will produce a very large search space. We tried to replicate the master plan using a 20 x 20 grid but the results produced a poor approximation. In addition, the design vector was still large with 400 variables. Scalability is always an issue with one to one mapping between genotype and phenotype.

We therefore adopted the concept of *Embryogeny* (Bentley and Kumar, 1999), which abstracts parts of the design with a set of rules in order to reduce the dimensionality of the design space (figure 7.9). This approach allowed us to model (and we believe will ultimately optimize) the different zones of the Masdar building system.

Figure 7.9:

Synthesis Embryogeny rules for recreating the Masdar layout



As part of the embryogeny approach we first synthesize a city layout around the concept of a spine or "backbone" consisting of the light rail transit (LRT) system and the green park structure according to the design patterns implemented by the Foster design team such as the ones seen in figure 7.8.

Next, seeds of the major buildings and focal points such as the University and headquarters are placed along the spine. The cell data structure is allowed to grow and evolve using a horizontal growth functions (figure 7.10).

Next, seeds for various zones such as special economic zones and commercial zones are placed on the spine and in regards to the SEZ, close to the University. The cell data structure is once again allowed to grow and evolve using a horizontal growth functions.

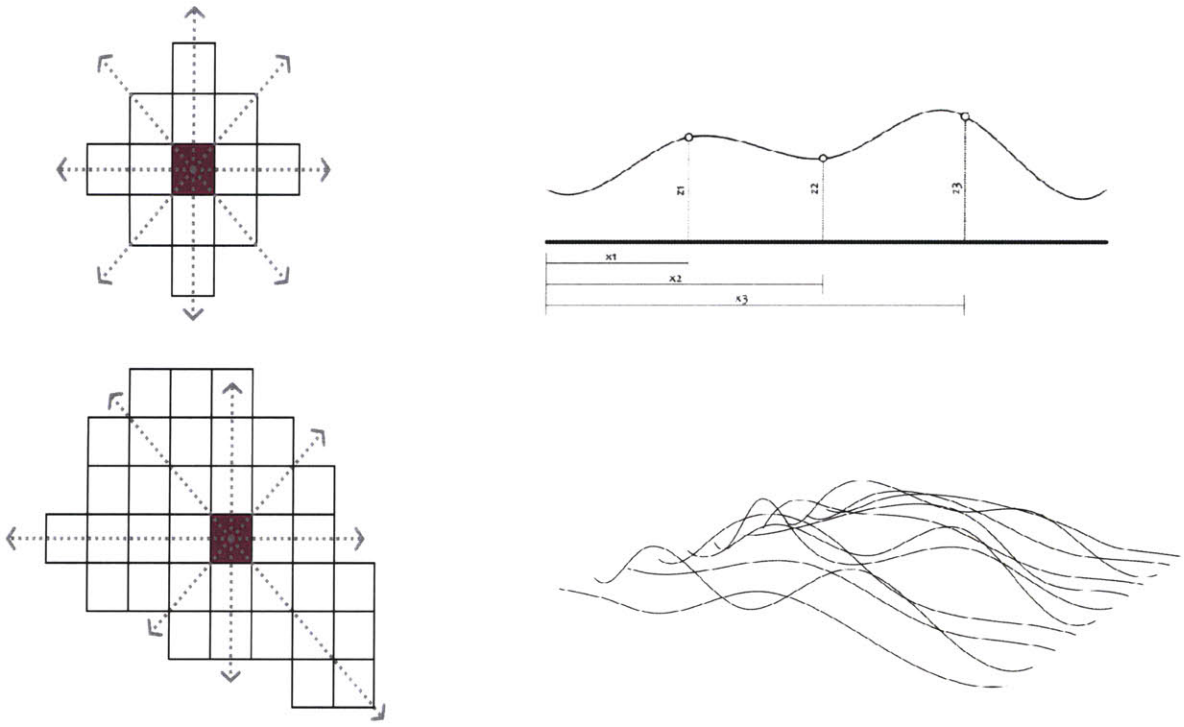
Next, the boundary zones which include the light industries and residential parking are placed and grown using a variation of the growth function. Finally the Residential zones fill out the rest of the city layout.

The city height is then controlled using a vertical growth function (*fcn-V*) that is driven by so called Profile Parameters. Figure 7.10 gives an example of how the horizontal growth function and the vertical growth function combine to achieve a final 3-dimensional city building system layout. We believe that with this approach we should be able to recreate not just the layout of the current proposal of Masdar, but many other historical, present and future cities as well.

Synthesis begins by specifying a set of design variables (DV), which can either be provided manually or are passed to the synthesis model as an output from the optimization model. This DV will include all the information that will drive the design algorithm specified above (figure 7.11).

Figure 7.10:

Synthesis horizontal and vertical growth functions



Growth *fnc* -H

Growth *fnc* -H

Growth *fnc* -V

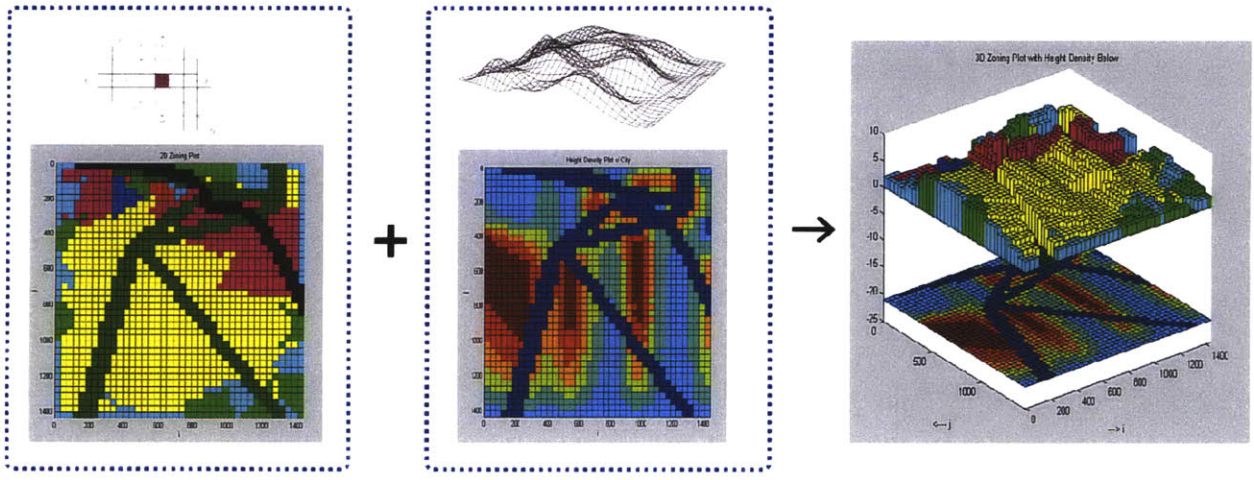
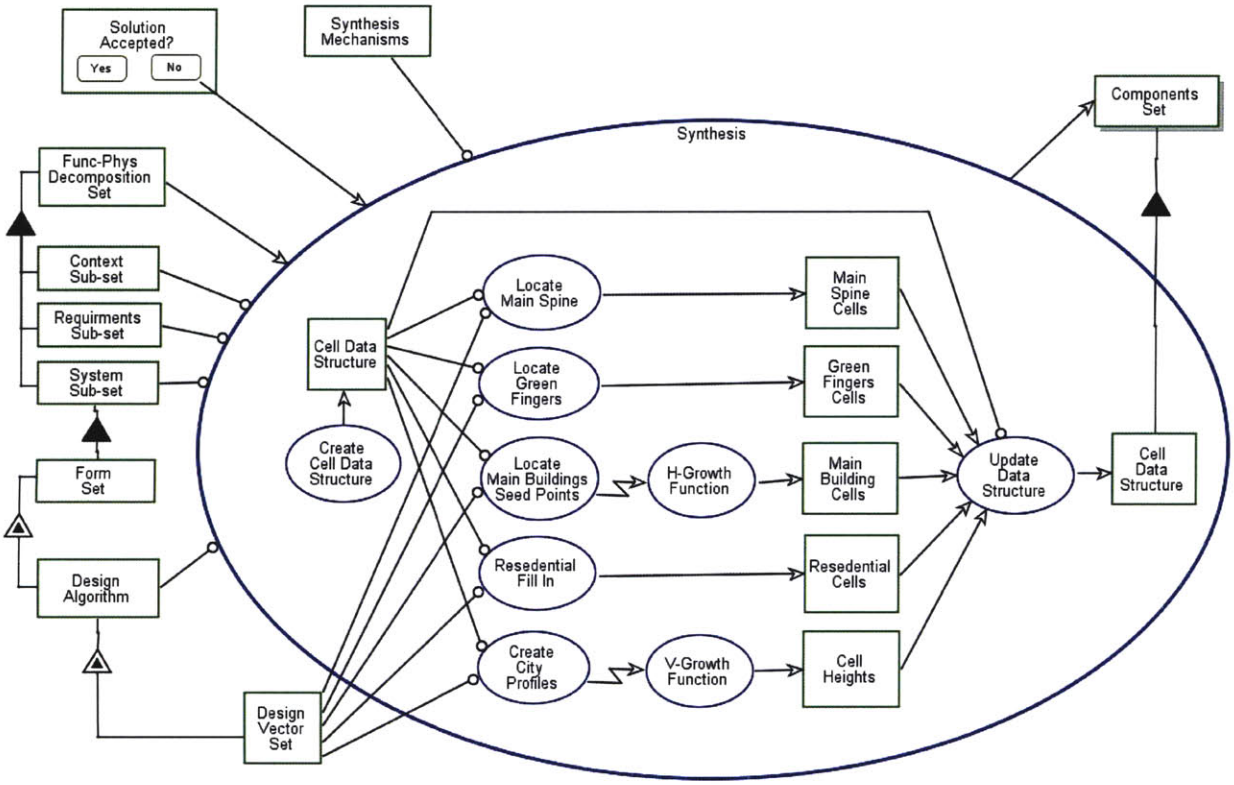


Figure 7.11:

Diagram of Synthesis Modules in Computational Design System



Analysis Models

Based on the objectives hierarchy three analysis modules were implemented within the design cycle to assess the behaviors of the synthesized building system. These models are the Area, Commuting and Energy modules (figure 7.12).

The cell structure serves as the basis of analysis. The properties of each cell are defined and used for analysis such as the area allocation for each zone, the distribution of commuting distances in the city or the distribution of energy usage and production.

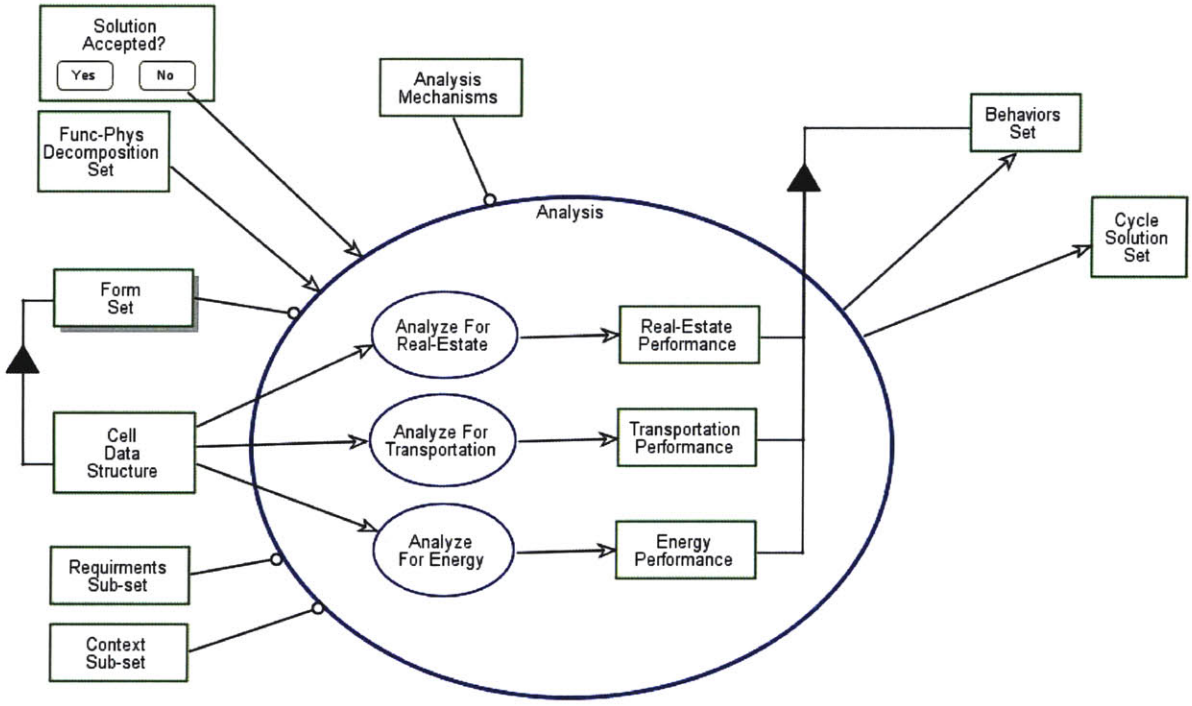


Figure 7.12:

Diagram of Analysis Modules in Computational Design System

Area Analysis Module

The total area of development is about 6 million square meters. The Special Economic Zones form about a quarter of the whole development in order to provide a substantial business and research district capable of attracting world leading companies.

The proximity of the Al Raha projects, Khalifa city and Yas Island will allow a population to commute from these areas using the light rail. Therefore only 30% of the site will be allocated for housing which will be largely composed of student accommodation. The university and civic uses will occupy much less space and will rely on quality developments for their success (Foster, 2007).

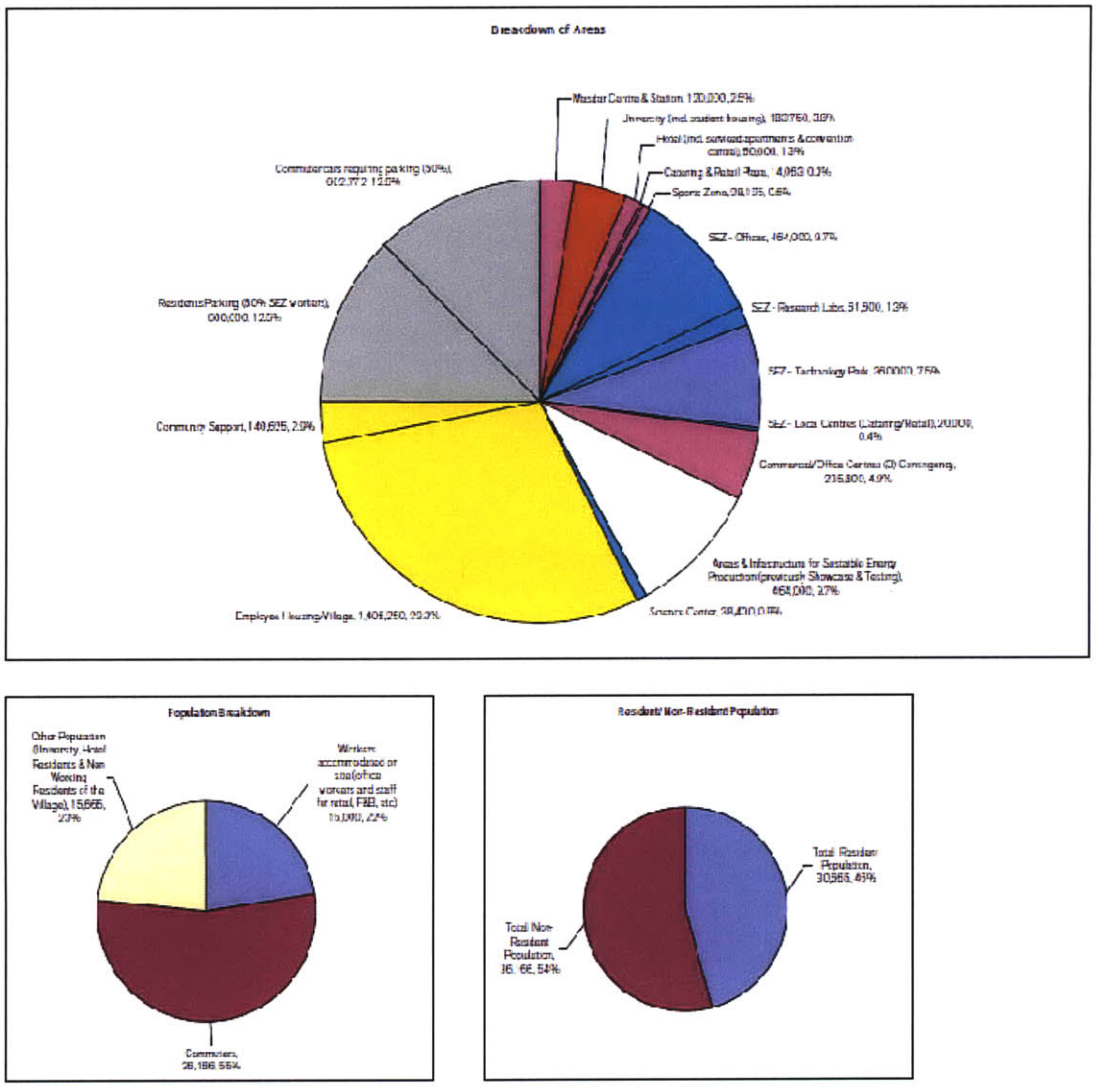


Figure 7.13:

Area Percentages breakdown (Foster, 2007)

Foster (2007) divides the percentages for the various uses breakdown as follows (figure 7.13):

University – 6%
SEZ – 24%
Civic and Culture – 8%
Commercial – 13%
Residential – 30%
Service and transport areas – 19%

Outside the walled zones additional area is allowed for the

following activities and facilities-
-Photovoltaic field
-Photovoltaic factory
-Desalination Plant
-Wind farm
-Research fields
-Various tree plantations of different species producing biofuel
-Water treatment Plant
-Visitors centre
-Visitors parking
-Recycling centre
-Sewage treatment plant
-Recreational areas
-Sports facilities

To calculate the actual area of each zone, the number of planar cells of the i^{th} zone type are summed then multiplied by the height of each cell. This area is then compared to the required area of each zone. The quality of the generated zoning area allocation is defined by the proximity of the actual generated area to required area of each zone.

$$A_{act,i} = a_{cell} * n_{act,i}$$

$$A_{req,i} = a_{cell} * n_{req,i}$$

$$J_{area,i} = min\left(\frac{A_{req,i}}{A_{act,i}}, \frac{A_{act,i}}{A_{req,i}}\right)$$

$$J_{area,i} \in (0; 1)$$

Average to obtain overall value:

$$J_{area} = \frac{1}{N}\sum_{i=1}^{N} J_{area,i}$$

$J_{area} = 1$ for optimal design

where:

A_{req} i^{th} area as per requirement
A_{act} i^{th} actual area
a_{cell} area of single cell = 1296sqm
$n_{act,i}$ number of actual cells generated by synthesis for i^{th} zone type
$n_{req,I}$ number of required cells for i^{th} zone type

Commuting Analysis Module

In the commuting analysis module, two types of commuting modes were implemented: commuting from the residential zone to the work region (zone) defined along the spine, and commuting from the residential zone to the recreation (parks) zone. Additional commuting types (type i) can be added in the future. Commuting from residential zones to any i^{th} zone type or region is analyzed by calculating the closest point distance to the i^{th} region of interest. For example, in the case of commuting to the spine region, for each residential cell, the closest point distance to the spine is calculated. To assess the quality of the design in regards to the commuting objective, quality value groups for each type are assigned based on how far a residential cell is from an i^{th} zone type of interest. An Example of the output is demonstrated in figure 7.14.

$$Q_{act,i} = \sum_{n=1}^{R} q_n$$

$$Q_{best,i} = R * q_{max}$$

$$J_{comm,i} = \frac{Qact,i}{Qbest,i}$$

$$J_{comm,i} \in (0;1)$$

Average to obtain overall value:

$$J_{comm} = \frac{1}{N}\sum_{i=1}^{N} J_{comm,i}$$

$J_{comm} = 1$ for optimal design

where:

$Q_{act,i}$ Commuting quality to i^{th} zone

$Q_{best,i}$ Best commuting quality value to i^{th} zone

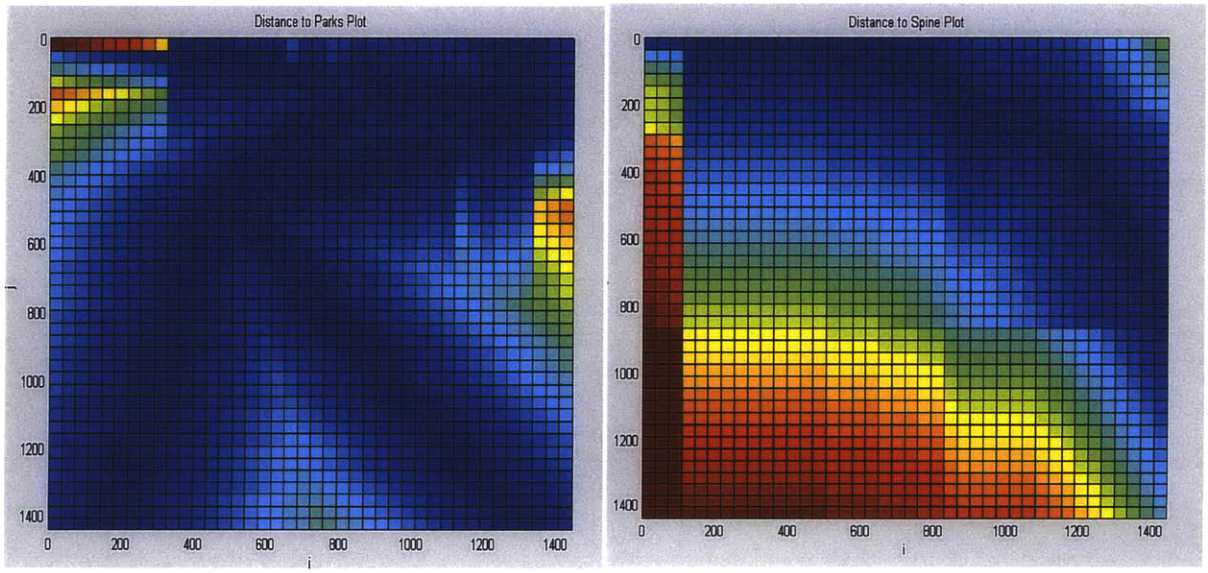
q_n Distance quality value group for n^{th} residential cell

R Number of residential cells

q_{max} Best distance quality value

Figure 7.14:

Plots of Commuting to Parks and Spine



Energy Analysis Module

To analyze the energy consumed by each zone, the rate of energy consumption per area per year is calculated then multiplied by the actual area of that zone (table 7.1). To analyze the energy generated by each generation method, the energy generated per area per year is calculated and multiplied by the

efficiency of that generation method. Any energy leakage, loss of generated energy, or impotence of captured energy caused by any reason is accounted for in the analysis by using assumed penalty factors. In the current analysis module, only solar energy produced by photovoltaics was assessed. Each zoning cell assumes a photovoltaic production on a rooftop at a conservative 15% efficiency. In addition, each cell checks neighbor cell heights to account for a shadowing penalty factor (5% per higher neighbor).

Table 7.1:

Rate of energy consumption per area per year per zone

Office Buildings (light and power) (central services)	485 MJ / m² yr 441 MJ / m² yr	Commercial, Parking Garages (central services only)
Public Buildings (Libraries, City Hall)	1102 MJ / m² yr	University
Laboratories	1131 MJ / m² yr	SEZ, Light Industries
Light Rail Transportation	10 kW-hr/mile, Avg 27126 mi / yr-car	LRT (assume 10 cars)
——	400 MJ / m² yr	Green Parks (based on central services)
Residential: Low Rise (24 units/acre) Mid Rise (48 units / acre) High Rise (96 units / acre)	 2344 MJ / m² yr (3.04 TJ / yr) 4152 MJ / m² yr (5.38 TJ / yr) 8580 MJ / m² yr (11.12 TJ / yr)	 (.02 ppl / m² OR 20 ppl / cell) (.03 ppl / m² OR 40 ppl / cell) (.06 ppl / m² OR 80 ppl / cell)

The city design is then evaluated with respect to the energy behavior by checking the proximity of the amount of energy generated to the amount of energy consumed where a design is assumed to be optimal when the two are equal. An Example of the output is demonstrated in figure 7.15.

$$E_{con} = \sum_{i=1}^{N} \left(e_{con,i} * A_{act,i} \right)$$

$$E_{gen} = \sum_{j=1}^{L} \left(e_{gen,j} * eff_{gen,j} * a_{cell} * \sum_{n=1}^{M} pf_{n,j} \right)$$

$J_{energy} = 1$ for optimal design

where:

$e_{con,i}$ Rate of energy consumed per area per year by i^{th} zone type

$e_{gen,j}$ Rate of energy generated per area per year by j^{th} energy generation method

$eff_{gen,j}$ Efficiency of j^{th} energy generation method

$pf_{n,j}$ penalty factor of j^{th} energy generation method for n^{th} zoning cell (equals 1 if there is no penalty, less than one otherwise)

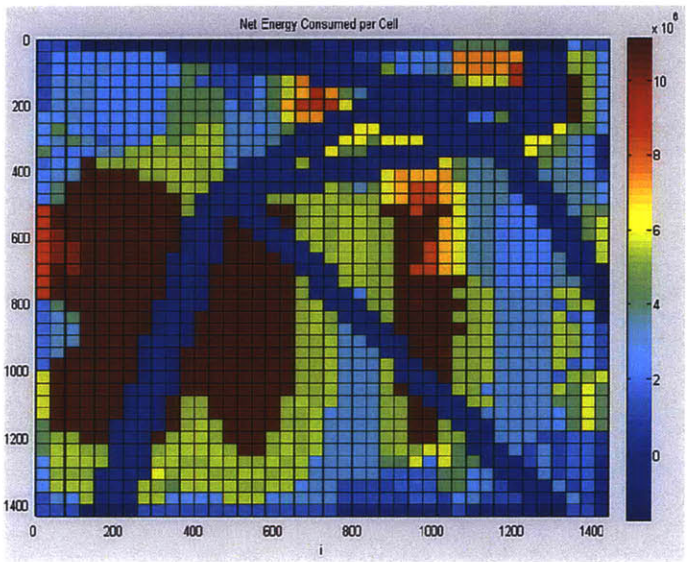
L Number of energy generation methods

M Number of zoning cells

N Number of zone types

Figure 7.15:

Plot of net energy consumed per cell



8. Conclusion

In this thesis I discussed several concepts to construct what I called the Evolutionary Design Model (EDM). As stated previously, the EDM embraces diverse areas of research that include design science, systems theory, and system modeling among many others.

I argue that a framework for developing an EDM would enhance the design of complex systems through an efficient process. This proposed framework would be generic and proposes a group of systematic methodologies that eventually lead to a fully realized and integrated design model.

The EDM is composed of several design states as well as design evolving processes. A design state describes a design at a particular point in time and maps the system's object to the system's requirements and identifies its relation to the context in which the system will operate. A design evolving process involves many sub-processes which include formulation, decomposition, modeling, and integration. These sub-processes are not always carried out in a sequential manner, but rather a continuous move back and forth to previous and subsequent stages is expected.

Design can be seen as an evolutionary process. One of the main aspects of the EDM framework is that it takes into account this evolutionary nature of design. As discussed earlier, design descriptions change as projects progress. A design cannot be described at the detailed level required for manufacturing at the earliest stages of design. The level of description of a specific design is directly proportional to the amount of

information available at a specific design state. With system design progress and evolution, the complexity of both the design description and the corresponding design models increase. Therefore, the resulting design model is described as an evolutionary model that moves a system's design from simple abstract states to more complex and detailed states throughout its evolution.

Although the design development process appears to be sequential with steps following each other, the reality is that certain knowledge can be gained or some circumstances can change as the process moves forward, thus questioning decisions early on in the process.

This notion of evolution yields an EDM that is continuously dependent on, and responsive to, the uncertainties of the design progress. New levels, new cycles as well as new models are added as the design progresses.

The type of model used within the EDM evolving processes is highly dependent on and driven by design needs of each process. As discussed earlier, the early EDM processes of Formulation and Decomposition implement more logical modeling Methods. Within the following phases of Modeling and Integration the focus is more on using mathematical modeling approaches. Furthermore, the EDM evolving processes can be implemented within a computational design system.

As the design evolves, models with different resolutions and granularity levels are needed. By altering models or exchanging existing models for more suitable fidelity levels, an existing design state evolve into a new design state. Therefore, EDM involves a multitude of model resolutions. In early design states, low-fidelity models are implemented due to the lack of complete and sufficient design information. In later design states, more detail is needed to perform elaborate synthesis

and analysis which requires higher-fidelity models.

As the design evolves coupled design cycles tend to decouple. Decoupling takes place both horizontally and vertically when dependencies between models, cycles or levels disappear. This happens when the various interconnected models are decomposed into different cycles which do not require as their input the output of another cycle. This decoupling of design cycles and levels can benefit from parallelism.

The EDM can also be described as an adaptive system. The number of modules and cycles needed at a particular design evolving process change depending on the available information.

Within this framework, complexities of the design can be handled and the uncertainty of its evolution can be managed. Furthermore, EDM processes provide a better understanding of not only the designed system attributes, but also of the priorities of the design system expectations and objectives.

The framework presented supports the design of complex systems within a variety of domains and the hope is that by implementing the EDM framework system engineers can enhance design quality and system performance.

Finally, a partial EDM was eventually implemented in a computational design system to help design Masdar city. The objective of the research is to complete the model fully to demonstrate the application of the proposed framework.

Bibliography

Abraham, A., Jain, L. and Goldberg, R. (Eds.) (2005). *Evolutionary Multi-Objective Optimization: Theoretical Advances and Applications.* Springer Science, New York, New York.

Alber, R. Rudolph, S. and Kršplin, B. (2002). "On Formal Languages in Design Generation and Evolution. Proc". 5th World Congress on Computational Mechanics (WCCM V), University of Vienna, Vienna, Austria.

Alexander, C. (1964). *Notes on the Synthesis of Form.* Harvard University Press, Cambridge, Massachusetts.

Alexander, C. (1964). *Notes on the Synthesis of Form.* Harvard University Press, Cambridge, Massachusetts.

Anderl, R. and Mendgen, R. (1996). "Modelling with constraints: theoretical foundation and application". *Computer-Aided Design.* 28(3). pp. 155-168.

Antoniou, A. Lu, W. Murray, W. and Wright, M. (2007). *Practical Optimization. Springer.* New York, New York.

Atherton, C. (2002). "An Approach to Multidisciplinary Design, Analysis & Optimization for Rapid Conceptual Design". AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization. AIAA, Atlanta Georgia.

Aughenbaugh, J. and Paredis, C. (2004). "The Role and Limitations of Modeling and Simulation in Systems Design". Proceedings of IMECE2004/2004 ASME International Mechanical Engineering Congress and RD&D Expo. Anaheim, California USA.

Averill, L. (2006). *Simulation Modeling and Analysis.* McGraw-Hill. New York, New York.

Bahrami, A. and Dagli, C. (1994). "Design Science". In: Dagli, C. and Kusiak, A (Eds.). *Intelligent Systems in Design and Manufacturing.* ASME Press, New York.

Baldwin, C. and Clark, K. (2000). *Design Rules, Vol. 1: The Power of Modularity.* MIT Press. Cambridge, MA.

Batty, M. (2005). *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals.* Cambridge, MA, MIT Press.

Bentley, P. and Kumar, S. (1999). "Three ways to grow designs: a comparison of embryogenies of an evolutionary design problem". In: Banzhaf, W. Daida, J. Eiben, A. Garzon, M. Honavar, V. Jakiela, M. and Smith, R. (Eds.). Genetic and Evolutionary Computation Conference, Orlando, FL, p. 35-43.

Bletzinger, K. and Lähr, A. (2006). "Prediction of interdisciplinary consequences for decisions in AEC design processes". ITcon. 11, Special Issue Process Modelling, *Process Management and Collaboration*, pp. 529-545

Buede, D. (2009). The Engineering Design of Systems: Models And Methods. Wiley Publishing. Hoboken, New Jersey.

Cagan J. (2001). "Engineering Shape Grammars". In: Antonsson E.K. and Cagan J. (Eds.). *Formal Engineering Design Synthesis*. Cambridge, Cambridge University Press.

Chapman, W., Bahill, A. and Wymore, A. (1992). Engineering Modeling and Design. Ann Arbor: CRC Press.

Chen, D. and Stroup, W. (1993). "General System Theory: Towards a Conceptual Framework for Science and Technology Education for All". Journal of Science Education and Technology 2(3), pp447-459.

Chomsky, N. (2002). *Syntactic Structures*. Walter de Gruyter. New York, New York.

Cook, H. (1997). *Product Management: Value, Quality, Cost, Price, Profits, and Organization*. Chapman & Hall.

Coyne, R. D. Rosenman, M. A. Radford, A. D. Balachandran, M. and Gero, J. S. (1990). *Knowledge-Based Design Systems*. Reading: Addison-Wesley.

Crawley, E. (2000). Lecture Notes for ESD.34 System Architecture. MIT, Cambridge, MA.

Crawley, E. (2003). Lecture Notes for ESD.34 System Architecture. MIT, Cambridge, MA.

Crawley, E. de Weck, O. Eppinger, S. Magee, C. Moses, J. Seeing, W. Schindall, J. Wallace, D. and Whitney, D. (2004). "The Influence of Architecture in Engineering Systems". In: *Engineering Systems Monographof the Engineering Systems Symposium*. MIT, Cambridge, MA.

Cross, N. (1989) *Engineering Design Methods*. Chichester, John Wiley & Sons Ltd.

Daffa', A. (1977). *The Muslim Contribution to Mathematics*. Croom Helm, London.

Dasgupta, S. (1989). "The Structure of Design Processes". In: Yovits M.C. (ed.). *Advances in Computers*.Academic Press. New York. 28, pp. 1-67.

Davis, K. and Bigelow, H. (2002). *Motivated metamodels:Synthesis of cause-effect reasoning and statistical modeling*, The RAND Corporation, Santa Monica,CA.

de Weck, O., Svetinovic, A., Dori, D., Alfaris, A., Syed, A., and Chepko, A. (2009). Requirements Engineering For Sustainable Systems. MIT/MIST Collaborative Research Progress Report for Period 9/1/08 to 7/11/09. Cambridge, MA.

De Weck, O. (2004). "Multi-Objective Optimization: History and Promise". Keynote Paper. The 3rd China-Japan-Korea Joint Symposium on Optimization of Structural and Mechanical Systems, Kanazawa, Japan.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. Chichester: John Wiley & Sons.

Dixon, J.R. (1987). *On Research Methodology towards a scientific theory of engineering design*. AI EDAM. 1 (3), pp. 145-157.

Dori, D. (2002). *Object-Process Methodology*. Springer-Verlag, New York.

Dori, D., Reinhartz-Berger, I., Sturm, A.(2003).OPCAT - A Bimodal CASE Tool for Object-Process Based System Development. Int. Conf, on Enterprise Information Systems, pp. 286-291.

Downing, F. and Flemming, U. (1981) "The Bungalows of Buffalo". *Environment and Planning* B. 8(3), pp. 269 – 293.

Duarte, J. (2001). "Customizing Mass Housing: a Discursive Grammar for Siza's Malagueira houses". Ph.D dissertation. Massachusetts Institute of Technology. Cambridge, Massachusetts.

Edgeworth, F.Y. (1881). *Mathematical Psychics*, P. Keagan, London, England.

Eggert, R.J. (2004). *Engineering Design*. Prentice Hall, Upper Saddle River, New Jersey.

Elster, K. (1993). *Modern Mathematical Methods of Optimization*. Wiley VCH. Berlin, Germany.

Eppinger, S. D. and Gebala, D. A. (1991) "Methods for Analyzing Design Procedures". *ASME Conference on Design Theory and Methodology. Miami, FL*. pp. 227-233.

Foster and Partners (2007). Abu Dhabi Future Energy Company Masdar Development Project, Detailed Masterplan, Master Plan Vol.1-4

Gero, J. S. (1990). "Design Prototypes: a Knowledge Representation Schema for Design". *AIMagazine*.11 (4): 26-36.

Gershenfeld, N. (1998). *Nature of Mathematical Modeling*. Cambridge University Press. New York, New York.

Grady, J. (1994). *System Integration*. CRC Press. Ann Arbor, Michigan.

Gries, M. (2004). "Methods for evaluating and covering the design space during early design development". Integration the VLSI Journal.38 (2), pp. 131-183.

Grobshtein, Y. and Dori, D. (2008). Evaluating Aspects of Systems Modeling Languages by Example: SysML and OPM.

Hastings, D. (2004). Lecture Notes for 16.892J / ESD.353J Space System Architecture and Design. MIT, Cambridge, MA.

Heisserman, J. Callahan, S. and Mattikalli, R. (2000). "A design representation to support automated design generation". In: Gero J. (ed). *Artificial Intelligence in Design*. pp 545-566.

Hemberg, M. (2001). Genr8 - a design tool for surface generation. Master's thesis, Chalmers University of Technology, Goteborg, Sweden.

Hongo, K. (1985). "On the significance of the theory of design". In: Yoshikawa, H. (Ed.). *Design and Synthesis*. Elsevier Science Publishers. Amsterdam.

Hornby, G. S., and Pollack, J. B. (2001a). "The Advantages of Generative Grammatical Encodings for Physical Design. Proceedings of the 2002 Congress on Evolutionary Computation.

Huebner, K. Dewhirst, D. Smith, D. Byrom, T. (2001). *The Finite Element Method for Engineers*. Wiley-IEEE. Hoboken, New Jersey.

Jacoby, S. and Kowalik, J. (1980). *Mathematical Modeling with Computers*. Prentice-Hall, Englewood Cliffs, New Jersey.

Kalay, Y. (1989). *Modeling Objects and Environments*. Wiley Publishing. Hoboken, New Jersey.

Kalay, Y. (2004). *Architecture's New Media: Principles, Theories, and Methods of Computer-Aided Design*. The MIT Press. Cambridge, Massachusetts.

Keeney, R. (1994). "Creativity in Decision Making with Value-Focused Thinking". Sloan Management Review. 35(4): pp. 33-41.

Koch, P. Evans, J. and Powell, D. (2002). "Interdigitation for effective design space exploration using iSIGHT". *Structural and Multidisciplinary*

Optimization. 23(2), 111–126.

Kockler, F. Withers, T. Poodiack, J. and Gierman, M. (1990). *Systems engineering management guide.* Defense Systems Management College. U.S. Government Printing Office. Washington, D.C.

Koning, H. *Eizenberg,* J. (*1981*). "The Language of the *Prairie*: Frank Lloyd *Wright's Prairie houses". Environment and Planning* B. 8(3), pp. 295 – 323.

Kossiakoff, A. and Sweet, W. (2002). *Systems Engineering Principles and Practice.* Wiley-Interscience, Hoboken, New Jersey.

Kroo, I.M. (1997a). "MDO for large-scale design". In: Alexandrov, N. M. and Hussaini M. Y. (Eds.), *Multidisciplinary Design Optimization: State of the Art.* SIAM, 1997, pp. 22-44.

Kuhn, T.S. (1970). *The Structure of Scientific Revolutions.* University of Chicago Press. Chicago, IL.

Law, A. and Kelton, W. D. (1999). *Simulation Modeling and Analysis.* McGraw-Hill, New York.

Locke, S. (2008) Middle East Business Intelligence (MEED). 02-17-2008. Retrieved 06-10-2009.

Maki, D. and Thompson, M.(2006). *Mathematical Modeling and Computer Simulation.* Thomson Brooks/Cole. Belmont, California.

McManus, H. Hastings, and D. Warmkessel, J. (2004). *Journal of Spacecraft and Rockets.*41(1), pp. 10-19.

Meredith, D.D., Wong, K.W., Woodhead, R.W. and Wortman, R.H. (1985). *Design and Planning of Engineering Systems.* Prentice-Hall, Englewood Cliffs, New Jersey.

Messac, A. (2000). "From Dubious Construction of Objective Functions to the Application of Physical Programming". *AIAA Journal,* 38(1), pp. 155-163.

Mitchell, W. (1990). *The Logic of Architecture: Design, Computation, and Cognition.* MIT Press. Cambridge, MA.

Nocedal, J. and Wright, S. (2000). *Numerical Optimization.* Springer. New York, New York.

OMG (2007a) *OMG Systems Modeling Language* (OMG SysML™) website: *http://www.omgsysml.org/.*

OMG (2007b) *OMG Systems Modeling Language* (OMG SysML™) Specification. SysML 1.0 Proposed Available Specification (PAS), OMG document [ptc/2007-02-03] dated 2007-03-23.

Open Group Architectural Framework (2001). www.opengroup.org.

Pahl, G. and Beitz, W. (1991). Engineering Design. Springer-Verlag. Berlin.

Pahng, F., Senin, N. and Wallace, D. (1997). "Modeling and evaluation of product design problems in a distributed design environment". Proceedings of the 1997 ASME Design Engineering Technical Conferences, Sacramento, California, September 14–17, DETC97/DFM-4356, CD-ROM (New York: ASME).

Papalambros, P. (2000). "Extending the Optimization Paradigm in Engineering Design". Proceedings of the 3rd International Symposium on Tools and Methods of Competitive Engineering, Delft, The Netherlands.

Papalambros, P. and Wilde, D. (2000). *Principles of Optimal Design.* Cambridge University Press. New York, New York.

Parry, G. (1996). "The Characterization of Uncertainty in Probabilistic Risk Assessment of Complex Systems". Reliability Engineering and System Safety, 54(2-3), pp. 119-126.

Pareto, V. (1906). *Manuale di Economia Politica*, Societa Editrice Libraria, Milano, Italy. Translated into English by A.S. Schwier, (1971). As *Manual of Political Economy*, Macmillan, New York.

Paydarfar, S. (2001). "An Integration Maturity Model for the Digital Enterprise". *The Digital Enterprise.* pp. 29-44.

Perry, D.E. and Wolf, A.L. (1992). *Foundations for the Study of Software Architecture.* ACM SIGSOFT. 17(4), pp. 40-52.

Prusinkiewics, P. and Lindenmayer, A. (1991) *The Algorithmic Beauty of Plants.* Springer-Verlag.

rawley, E. de Weck, O. Eppinger, S. Magee, C. Moses, J. Seeing, W. Schindall, J. Wallace, D. and Whitney, D. (2004). "The Influence of Architecture in Engineering Systems". In: *Engineering Systems Monographof the Engineering Systems Symposium.* MIT, Cambridge, MA.

Requicha, A. (1980). "Representations of Rigid Solids: Theory, Methods and Systems". *ACM Computing Surveys.* 12(4), pp. 437-466.

Reynolds, F. Natrajan, A., Srinivasan S. (1997). "Consistency maintenance in multiresolution simulation". *ACM Transactions on Modeling and Simulation,* 7(3) pp. 368-392.

Rosenman, M. and Simoff, S. (2001). "Some conceptual issues in component-assembly modeling". *Artificial Intelligence in Engineering.* 15(2), pp. 109-119.

Sacks, R. Eastman, C. Lee, G. (2004). "Parametric 3D Modeling in Building Construction with Examples from Precast Concrete". *Automation in Construction*. 13, pp 291– 312.

Saff, E. and Snider, A. (1993). *Fundamentals of Complex Analysis for Mathematics, Science, and Engineering*. Prentice Hall.

Sage, A.P. and Armstrong Jr., J.E. (2000). *Introduction to Systems Engineering*. Wiley-Interscience. Malden, Massachusetts.

Schmidt, J.W. and Taylor, R.E. (1970). *Simulation and Analysis of Industrial Systems*. Richard D. Irwin, Homewood, Illinois.

Shah, J. and MŠntylŠ, M. (1995).*Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. Wiley Publishing. Hoboken, New Jersey.

Simon, H. A. (1973). *The Sciences of the Artificial*. MIT Press. Cambridge, MA.

Smith R., Eppinger S. (1997). "Identifying Controlling Features of Engineering Design Iteration". *Management Science*. 43(3), March, pp. 276-293.

Stevens, R. and Martin, J. (1995). What is requirements Management? In Kirkpatrick, C. and Wilke C. (Eds.). Systems Engineering in the Global Market Place Vol.2 . 5th annual International symposium of INCOSE, 11-32.

Stiny, G. and Gips, J. (1972). *Shape Grammars and the Generative Specification of Painting and Sculpture*. C V Freiman (Ed) Information Processing 7. Amsterdam, North–Holland, 1460–1465.

Stiny, G. and Mitchell, W. J. (1978) "The Palladian Grammar". *Environment and Planning* B 5: 5-18.

Suh, N.P. (1990). *The Principles of Design*. Oxford University Press. New York.

Reinhartz-Berger, I. and Dori, D. (2004). Object-Process Methodology (OPM) vs. UML: A Code Generation Perspective.

Trefethen, L. and Bau, D. (1997). *Numerical Linear Algebra*. SIAM, Philadelphia, Pennsylvania.

Tyng, A. (1984) *Beginnings*. Wiley. New York.

Ulrich, K. (1995). "The Role of Product Architecture in the Manufacturing Firm". *Research Policy*. 24(3), pp. 419-440.

Ulrich, K. and Seering, W.P. (1990). "Function Sharing in Mechanical Design". *Design Studies*. 11(4), pp. 223-234.

Ulrich, K. and Seering, W.P. (1990). "Function Sharing in Mechanical Design". *Design Studies*. 11(4), pp. 223-234.

Ulrich, K.T. and Ellison, D. J. (1999) "Holistic Customer Requirements and the Design-Select Decision". *Management Science*. 45(5), pp. 641-658.

Ulrich, K.T., and Eppinger, S.D. (2000). *Product Design and Development*. McGraw-Hill. New York. New York.

Wand, W. and Weber R. (1989) On the Ontological Evaluation of Information Systems Analysis and Design Methods. In Falkenberg, E. and Lindgreeen, P. (Eds.) *Information System Concepts: An In-Depth Analysis*. Elsevier Science Publishers.

Whitney, D.E. (1996). "Why Mechanical Design Cannot Be Like VLSI Design". *Research in Engineering Design*.8 (3), pp. 125- 138.

Whitney, D.E. (1996). "Why Mechanical Design Cannot Be Like VLSI Design". *Research in Engineering Design*.8 (3), pp. 125- 138.

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media. Champaign, *Illinois.*

Yilmaz, L. and Ören, T.I. (2004). "Dynamic Model Updating in Simulation with Multi-models: A Taxonomy and a Generic Agent-Based Architecture". *Proceedings of SCSC 2004 - Summer Computer Simulation Conference, San Jose, CA*. pp. 3-8.

Zachman, J.A. (1987). "A framework for information systems architecture". *IBM Systems Journal*. 26(3), pp. 276-292.

Zeigler, B. P., Praehofer, H. Kim, G. T. (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

Figures List

Tables List